PMAA-00077-US

日 本 国 特 許 庁
PATENT OFFICE
JAPANESE GOVERNMENT

J1036 U.S. PTO
09/777954
02/07/01

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

#4
Priority Papers
4/26/01

出 願 年 月 日
Date of Application: ２０００年 ９月２２日

出 願 番 号
Application Number: 特願２０００－２８９３７３

出 願 人
Applicant (s): 三菱電機株式会社

CERTIFIED COPY OF
PRIORITY DOCUMENT

２０００年１０月１３日

特 許 庁 長 官
Commissioner,
Patent Office 及 川 耕 造

出証番号 出証特２０００－３０８４７４６

【書類名】　特許願

【整理番号】　526014JP01

【提出日】　平成12年 9月22日

【あて先】　特許庁長官殿

【国際特許分類】　G05B 19/05

G06F 9/06

【発明者】

【住所又は居所】　東京都千代田区丸の内二丁目２番３号　三菱電機株式会社内

【氏名】　南出　英明

【発明者】

【住所又は居所】　東京都千代田区丸の内二丁目２番３号　三菱電機株式会社内

【氏名】　田中　輝明

【発明者】

【住所又は居所】　東京都千代田区丸の内二丁目２番３号　三菱電機株式会社内

【氏名】　塩本　佳子

【発明者】

【住所又は居所】　東京都千代田区丸の内二丁目２番３号　三菱電機株式会社内

【氏名】　南角　茂樹

【特許出願人】

【識別番号】　000006013

【氏名又は名称】　三菱電機株式会社

【代理人】

【識別番号】　100089118

【弁理士】

【氏名又は名称】　酒井　宏明

【手数料の表示】

【予納台帳番号】　036711

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9803092

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　プログラマブルコントローラおよび制御プログラム開発支援装置

【特許請求の範囲】

【請求項１】　制御プログラムをコンパイルすることで生成された実行コードに基づいてシーケンス処理をおこなうプログラマブルコントローラにおいて、

前記実行コードを記憶する記憶手段と、

パイプラインロジックやキャッシュ等の高速化実装手段を搭載するとともに、前記実行コードによって直接実行される汎用マイクロプロセッサと、

を備えたことを特徴とするプログラマブルコントローラ。

【請求項２】　ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、

前記制御プログラムを、パイプラインロジックやキャッシュ等の高速化実装手段を搭載した汎用マイクロプロセッサにおいて直接実行可能な実行コードにコンパイルするコンパイル手段を備えたことを特徴とする制御プログラム開発支援装置。

【請求項３】　ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、

前記制御プログラムを複数のブロックに分割する制御プログラム分割手段と、

前記複数のブロックのすべてまたは一部をプログラマブルコントローラにおいて直接実行可能な実行コードにコンパイルするコンパイル手段と、

を備えたことを特徴とする制御プログラム開発支援装置。

【請求項４】　ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、

前記制御プログラムを複数のブロックに分割する制御プログラム分割手段と、

前記複数のブロックのすべてまたは一部を、当該ブロック毎に、汎用コンピュ

ータ用高級言語で記述された高級言語制御プログラムに変換する制御プログラム変換手段と、

前記ブロック毎に対応する汎用コンピュータ用高級プログラミング言語のすべてまたは一部をプログラマブルコントローラにおいて直接実行可能な実行コードにコンパイルするコンパイル手段と、

を備えたことを特徴とする制御プログラム開発支援装置。

【請求項５】　前記プログラマブルコントローラは、パイプラインロジックやキャッシュ等の高速化実装手段を搭載する汎用マイクロプロセッサを備えたことを特徴とする請求項３または４に記載の制御プログラム開発支援装置。

【請求項６】　前記制御プログラムは、ラダーダイヤグラムまたはラダーダイヤグラムから生成されたインストラクションリストであり、

前記制御プログラム分割手段は、前記ラダーダイヤグラム内の所定のラングにおいて複数のブロックに分割して、当該ブロック毎にプログラムファイルを生成することを特徴とする請求項３、４または５に記載の制御プログラム開発支援装置。

【請求項７】　前記制御プログラムは、ラダーダイヤグラムまたはラダーダイヤグラムから生成されたインストラクションリストであり、

前記制御プログラム分割手段は、前記ラダーダイヤグラム内のジャンプ命令に対するジャンプ先となる所定のラングにおいて複数のブロックに分割し、当該ブロック毎にプログラムファイルを生成することを特徴とする請求項３、４または５に記載の制御プログラム開発支援装置。

【請求項８】　前記制御プログラムは、ラダーダイヤグラムまたはラダーダイヤグラムから生成されたインストラクションリストであり、

前記制御プログラム分割手段は、前記ラダーダイヤグラムにおいて共通の入力デバイスまたは出力デバイスに対する命令を含んだラングのすべてまたは一部を抽出し、抽出したラングのすべてまたは一部により一つのブロックを構成するとともに当該ブロック毎にプログラムファイルを生成することを特徴とする請求項３、４または５に記載の制御プログラム開発支援装置。

【請求項９】　前記制御プログラムを、参照されない変数の排除、冗長コー

ドの排除、共通の入力デバイスや出力デバイスに対する命令を局所的に配置するためのコードの並べ替え等をおこなうことによって最適なコード体系に再構築する最適化フィルタ手段を備え、

前記最適化フィルタ手段により最適化された制御プログラムをあらためて前記制御プログラムとすることを特徴とする請求項２～８のいずれか一つに記載の制御プログラム開発支援装置。

【請求項１０】　処理時間が既知であるサンプルプログラムと前記実行コードに対応する制御プログラムとを対応付けた対応表を有し、当該対応表に基づいてプログラマブルコントローラにおけるシーケンス処理実行時間を見積もる処理時間概算手段を備えたことを特徴とする請求項２～９のいずれか一つに記載の制御プログラム開発支援装置。

【請求項１１】　ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、

前記制御プログラムを汎用コンピュータ用高級プログラミング言語で記述された高級プログラミング言語制御プログラムに変換する制御プログラム変換手段と、

前記高級プログラミング言語制御プログラムを構成するソースコード内において、前記インストラクションリストを構成する各行に対応した部分に当該行番号を示すコードを挿入してデバッグ用制御プログラムを生成するデバッグコード生成手段と、

前記デバッグ用制御プログラムをステップ実行することにより前記インストラクションリストの各行と前記高級プログラミング言語制御プログラムの実行部分とを対応付けて表示するデバッグ実行手段と、

を備えたことを特徴とする制御プログラム開発支援装置。

【請求項１２】　制御プログラムをコンパイルすることで生成された実行コードに基づいてシーケンス処理をおこなうプログラマブルコントローラにおいて、

前記実行コードを記憶する第１の記憶手段と、

前記第１の記憶手段に記憶されている実行コードと新たな実行コードとの差分データを記憶する第２の記憶手段と、

前記実行コードにより直接実行されるマイクロプロセッサと、

前記差分データに基づいて、所定のタイミングで、現在実行中の実行コードを新たな実行コードに変更するとともに、変更後の実行コードを継続的に実行させるパッチ処理手段と、

を備えたことを特徴とするプログラマブルコントローラ。

【請求項１３】　制御プログラムをコンパイルすることで生成された実行コードに基づいてシーケンス処理をおこなうプログラマブルコントローラにおいて、

前記実行コードを記憶する記憶手段と、

前記実行コードにより直接実行されるマイクロプロセッサと、

を備え、

前記実行コードは、前記制御プログラムの圧縮により生成されたバイナリデータを含むことを特徴とするプログラマブルコントローラ。

【請求項１４】　ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、

前記制御プログラムを圧縮して圧縮ファイルを生成する圧縮手段と、

前記圧縮ファイルを前記制御プログラムのコード体系に変換した圧縮データを生成するコード変換手段と、

前記制御プログラムと前記圧縮データとを結合するとともに、当該結合結果を、プログラマブルコントローラにおいて直接実行可能な実行コードにコンパイルするコンパイル手段と、

を備えたことを特徴とする制御プログラム開発支援装置。

【請求項１５】　ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムに基づいて、シーケンス処理をおこなうプログラマブルコントローラにおいて、

前記制御プログラムを記憶する記憶手段と、

前記制御プログラムに使用されている各命令の出現頻度をカウントする命令カウント手段と、

前記命令カウント手段によってカウントされた結果から、出現頻度の高い順に命令を列挙したパターンマッチング表を作成するパターンマッチング表作成手段と、

前記制御プログラムを、前記パターンマッチング表に列挙されている命令順にパターンマッチングしつつ、前記プログラマブルコントローラにおいて直接実行可能な実行コードにインタプリーティングしつつ実行するインタプリーティング手段と、

を備えたことを特徴とするプログラマブルコントローラ。

【請求項１６】　ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、

前記制御プログラムに使用されている各命令の出現頻度をカウントする命令カウント手段と、

前記命令カウント手段によってカウントされた結果から、出現頻度の高い順に命令を列挙したパターンマッチング表を作成するパターンマッチング表作成手段と、

前記制御プログラムを、前記パターンマッチング表に列挙されている命令順にパターンマッチングしつつ、前記プログラマブルコントローラにおいて直接実行可能な実行コードにコンパイルするコンパイル手段と、

を備えたことを特徴とする制御プログラム開発支援装置。

【請求項１７】　前記請求項２～１１、１４および１６のいずれか一つに記載の制御プログラム開発支援装置を備えたことを特徴とする請求項１、１２、１３または１５に記載のプログラマブルコントローラ。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

この発明は、インストラクションリストやラダーダイヤグラム等の制御プログ

ラムをコンパイルすることで生成された実行コードに基づいてシーケンス処理をおこなうプログラマブルコントローラと上記制御プログラムのコンパイルやデバッグ等をおこなう制御プログラム開発支援装置に関するものである。

【０００２】

【従来の技術】

産業界の現場、特に生産現場においては規模の大小に関わらず生産設備や生産方式が激変してきており、生産のための設備や装置における電気系制御機器の範囲も年々増加してきている。そこで、そのような環境の中で、如何に機械系と連携を取り、如何に合理的にコストパフォーマンスを高め、高速にかつ取り扱いやすいユーザインターフェースを実現させるかが重要となっている。

【０００３】

これを成功させる手法の一つとして、プログラマブルコントローラ（ＰＣ）およびそれを制御するための開発環境が実用化されている。プログラマブルコントローラは、プログラマブルロジックコントローラ（ＰＬＣ）やシーケンサとも呼ばれており、一般に、シーケンス処理用の特別な制御プログラムの記述にしたがって動作する。

【０００４】

シーケンス処理は、各種センサや各種スイッチ等の入力デバイスから信号を入力し、入力した信号の状態に応じて、または特定の順番によって、表示機器や電磁リレー等の出力デバイスに対して制御信号を送出することを基本的な流れとしている。すなわち、制御プログラムは、このシーケンス処理の流れを記述したものである。

【０００５】

一般に、この制御プログラムは、パーソナルコンピュータ等の汎用コンピュータやプログラムローダ（以下、制御プログラム開発支援装置と称する。）で開発され、プログラマブルコントローラへは、通信インターフェースを介して転送される。

【０００６】

制御プログラムを記述する言語のうち代表的なものとして、ラダーダイヤグラ

ム方式、ファンクションブロックダイアグラム方式、構造化言語（Ｓｔｒｕｃｔｕｒｅｄ　Ｔｅｘｔ）方式およびＳＦＣ（Ｓｅｑｕｅｎｔｉａｌ　Ｆｕｎｃｔｉｏｎ　Ｃｈａｒｔ）方式が知られている。特に、ラダーダイヤグラム方式は、回路図がはしご状に描かれ、両端の垂直な二つの線が動力をシンボリックに表わし、その間で水平に結ぶ平行な線上にリレー接点回路の動力の流れが示される方式であり、最も広く使用されている言語である。

【０００７】

このラダーダイヤグラム方式では、制御プログラム開発支援装置を用いて、まず、上記したように人間が直感的に理解しやすい図記号の配置によってシーケンス処理の流れを表現し、つづいて、制御プログラム開発支援装置のコンバータ等により、この図記号の配置を、プログラマブルコントローラが解釈できるニーモニックコードで表したインストラクションリストに変換する。

【０００８】

よって、プログラマブルコントローラは、このインストラクションリストを受信し、そのインストラクションリストが示す命令にしたがったシーケンス処理を実行する形態を採用している場合がある。この場合、プログラマブルコントローラは、受信したインストラクションリストを一行ずつインタプリーティングすること、すなわちプログラマブルコントローラに搭載されたマイクロプロセッサにおいて直接実行可能なコードに変換することで逐次的に制御プログラムを実行する。以下において、このプログラマブルコントローラをインタプリタ型プログラマブルコントローラと称する。

【０００９】

また、他のプログラマブルコントローラとして、プログラマブルコントローラに搭載されたマイクロプロセッサにおいて直接実行可能な実行コードを、制御プログラム開発支援装置から受信して実行するコンパイル型プログラマブルコントローラも実用化されている。このコンパイル型プログラマブルコントローラを利用する形態では、制御プログラム開発支援装置にコンパイラを備えており、制御プログラム開発支援装置は、そのコンパイラによってラダーダイヤグラムを直接、または一旦インストラクションリストに変換した後、上記マイクロプロセッサ

において直接実行可能な実行コードにコンパイルし、その実行コードをプログラマブルコントローラに転送する。

【００１０】

ところが、上記した制御プログラムは、プログラマブルコントローラのベンダーや機種に依存しており、同じシーケンス処理を、同一のソースコードの制御プログラムを使用して、異なる機種のプログラマブルコントローラ間で実行することはできなかった。そのため、制御プログラムの新たな開発や新機種または異機種のプログラマブルコントローラへの移植作業が必要となり、過去の制御プログラムの資産を活用することが困難であった。

【００１１】

そこで、制御プログラムを、ラダーダイヤグラムやインストラクションリスト等の従来のシーケンス処理用の言語ではなく、パーソナルコンピュータで用いられる高級プログラミング言語で記述し、その高級プログラミング言語をコンパイラによってコンパイルすることにより実行コードを得る方法が提案されている。

【００１２】

例えば、特開平７－２９５６１２号に開示の「高級言語シーケンス命令プログラム生成装置および汎用シーケンサ」によれば、高級言語シーケンス命令プログラム生成装置が、翻訳プログラムによってシーケンス命令言語シーケンス命令プログラムを高級言語シーケンス命令プログラムに翻訳し、高級言語シーケンス命令プログラム生成装置が、その高級言語シーケンス命令プログラムをコンパイルして実行形式シーケンス命令プログラム生成する。または、汎用シーケンサが、上記高級言語シーケンス命令プログラムを順にインタプリーティングしながら実行する。これにより、シーケンサのベンダーや機種を問わず、同じ制御プログラムによって、シーケンス制御を実行することができる。

【００１３】

また、プログラムコントローラに記憶された制御プログラムの内容を更新する方法として、二つの方法が知られている。一つは、プログラマブルコントローラに新規に制御プログラムを転送する場合と同様の手順で、制御プログラム開発支援装置が、停止状態にあるプログラマブルコントローラに対して制御プログラム

を転送し、プログラマブルコントローラはその制御プログラムの受信が完了した後に、実行処理に移行するという方法である。

【００１４】

もう一つは、プログラマブルコントローラは、実行中の制御プログラムの内容を、実行状態を継続しつつ新しい制御プログラムの内容に置き換える方法（以下、ＲＵＮ中書き込み方法と称する。）である。通常、プログラマブルコントローラは、実行すべき各命令に一対一に対応したバイナリコード列のループを繰り返しインタプリーティングして実行することで動作しているが、この方法は、まず、空いているメモリ空間に新しい制御プログラムを読み込み、読み込み完了後、上記ループの開始または終了タイミングにおいて、元の制御プログラムの実行を停止するとともに新しい制御プログラムの実行を開始することで実現される。

【００１５】

なお、上記した制御プログラム開発支援装置によって実現される制御プログラムの開発やデバッギングと、プログラマブルコントローラのシーケンス処理とを一つの装置で実現した、いわゆるソフトウェアプログラマブルコントローラも実用化されている。このソフトウェアプログラマブルコントローラの装置構成は、汎用コンピュータの装置構成と同等であり、外部の入力デバイスおよび出力デバイスに対するシーケンス処理をソフトウェア的に実現している。

【００１６】

【発明が解決しようとする課題】

しかしながら、上述したような従来のプログラマブルコントローラおよび制御プログラム開発支援装置には、以下の問題点がある。

【００１７】

第１に、近年においてパーソナルコンピュータ等の汎用コンピュータに搭載されるマイクロプロセッサ（以下、汎用マイクロプロセッサと称する。）が高機能かつ高速に動作し、また安価に入手できることから、従来のプログラマブルコントローラに搭載されたマイクロプロセッサをＡＳＩＣ（Ａｐｐｌｉｃａｔｉｏｎ Ｓｐｅｃｉｆｉｃ ＩＣ）によって開発する利点が薄れつつあるという問題がある。

【００１８】

特に、汎用マイクロプロセッサでは、１次キャッシュおよび２次キャッシュを備えるとともに、パイプライン処理、スーパスケーラ処理およびアウトオブオーダ処理等の高速化実装技術によって構築されており、その汎用マイクロプロセッサにおいて直接実行可能な実行コードもまた、上記高速化実装技術を最大限に生かして最適なコード体系となるような実行コードに変換するコンパイラによって生成されている。

【００１９】

このような汎用コンピュータのコンパイラの多くは、ユーザが理解しやすい高級プログラミング言語によって記述されたプログラムをコンパイルするので、そのようなコンパイラを含めて、プログラマブルコントローラに汎用プロセッサを搭載してシーケンス処理を実行させることは大きな利点がある。ところが、汎用コンピュータでは、プログラム開発の出発点を高級プログラミング言語の記述としているので、プログラマブルコントローラにおいて使用されてきたラダーダイヤグラムやインストラクションリスト等の制御プログラム、すなわちシーケンス処理の過去の資産を活用することはできないという問題が生じてしまう。

【００２０】

第２に、入力デバイスと出力デバイスの多様化およびシーケンス処理の複雑化にともない、制御プログラムが長大化する傾向にあり、その開発、コンパイル、インタプリーティングおよびデバッギングに多くの時間を要しているという問題がある。

【００２１】

第３に、コンパイル型のプログラマブルコントローラでは、実行コードを受け取って直接シーケンス処理に移行するため、元の制御プログラムの各命令単位または処理ブロック単位に対応する実際の処理時間を見積もることが困難であるという問題がある。

【００２２】

第４に、上記した特開平７－２９５６１２号に開示の「高級言語シーケンス命令プログラム生成装置および汎用シーケンサ」は、インストラクションリスト等

の制御プログラムで記述されたソースファイルを例えばＣ言語のような高級プログラミング言語のソースファイルに変換してオブジェクトファイルを生成するが、その高級プログラミング言語のソースファイルのみを修正した場合、元の制御プログラムのソースファイルは、その修正によって変化しないため、二つの同じシーケンス処理を示すソースファイル間において整合がとれないという問題がある。

【００２３】

特に、制御プログラムを高級プログラミング言語で開発することのできる制御プログラム開発支援装置において、例えばデバッギングツールを用いてステップ実行をおこなおうとする場合、高級プログラミング言語での１行としてはステップ実行可能ではあるが、高級プログラミング言語に変換される前のインストラクションリスト等の言語形態に対しては、１行ごとのステップ実行をおこなうことができない。

【００２４】

また、一般的に、プログラマブルコントローラが稼動している現場では、高級プログラミング言語を直接操作できる制御プログラム開発支援装置を設置していないことが多く、シーケンス処理に不具合が生じた場合や制御プログラムの簡単な修正が生じた場合には、インストラクションリスト等のシーケンス処理用の言語のみが利用可能な簡易な制御プログラム開発支援装置を持ち込んで、デバッグ作業をおこなうことがある。すなわち、この場合には、上記問題が顕著となる。これに対応するためには、すべての現場において、高級プログラミング言語を直接操作できる制御プログラム開発支援装置を準備する必要があり、それにかかるコストが多大となってしまう。

【００２５】

第５に、上記したＲＵＮ中書き込み方法の実現は、新たな制御プログラムを読み込むために、現在実行中である制御プログラムが読み込まれているメモリと同等なサイズを有するメモリを別途用意しておかなければならないという問題がある。一般的にハードウェア部品においてメモリは高価なため、このＲＵＮ中書き込み方法を実現するためには、メーカまたはユーザが多くの投資を支払わなけれ

ばならない。

【００２６】

第６に、上記したコンパイル型の制御プログラム開発支援装置においては、一般に、最終的に生成した実行コードから元のインストラクションリスト等の制御プログラムを復元することは不可能であるという問題がある。そのため、例えば、プログラマブルコントローラと制御プログラム開発支援装置とが別筐体であり、かつプログラマブルコントローラと制御プログラム開発支援装置とがオフラインで隔離されている場合には、元の制御プログラムと実際に実行されている実行コードとは別場所に保管されることになる。

【００２７】

特に、コンパイル型の制御プログラム開発支援装置においては、その制御プログラム開発支援装置側で制御プログラムのソースファイルを消失した場合には、そのインストラクションリスト等の制御プログラムを復元することはできずに、デバッグ作業が不可能となり、その場合、制御プログラムを一から書き直さなければならなかった。

【００２８】

また、制御プログラムのソースファイルを消失していない場合であっても、現場にて動作中のプログラマブルコントローラの処理内容を変更する場合には、元の制御プログラムのソースファイルを保存している制御プログラム開発支援装置にアクセスし、そのファイルを取り出さなければならないという煩雑さがあった。

【００２９】

第７に、上記したプログラマブルコントローラまたは制御プログラム開発支援装置は、インタプリタ型であれ、コンパイル型であれ、ユーザが記述した制御プログラムの処理内容を逐次解釈する部分が存在するが、処理内容の解釈手順は必ずしも効率的とはいえないという問題がある。その逐次解釈をおこなう部分（以下、命令解釈手段と称する。）は、インタプリタ型のプログラマブルコントローラでは、実行時に制御プログラムを構成する各データを読み込み、それがどの命令の実行を指定しているかを解釈する部分であり、コンパイル型の制御プログラ

ム開発支援装置では、制御プログラムを逐次読み込んで解釈し、それを適当な出力フォーマットに変換する部分である。

【００３０】

命令解釈手段は、一般的に、入力されたデータに対し、解釈側で保有しているパターンとそのデータを比較し、それが適合（マッチ）した場合に、あらかじめ埋め込まれた動作（アクション）を実行する構造で記述されている。従って、この命令解釈手段のパターンマッチングの時間は、インタプリタ型ではその実行時、コンパイル型ではコンパイル時に多くの時間が必要となり、両者とも生産性を阻害する要因になっている。

【００３１】

この発明は上記問題点を解決するためになされたもので、第１に、より高速なシーケンス処理の実現と安価に入手できるという利点を有する汎用マイクロプロセッサを搭載したプログラマブルコントローラと、インストラクションリスト等の制御プログラムから、汎用マイクロプロセッサにおいて直接実行可能な実行コードを生成する制御プログラム開発支援装置とを得ることを目的とする。

【００３２】

第２に、インストラクションリスト等の制御プログラムを、所定の位置で分割し、分割したブロックごとにファイルを生成することで、再コンパイル等をおこなう部分を最小限に抑えることができる制御プログラム開発支援装置またはプログラマブルコントローラを得ることを目的とする。

【００３３】

第３に、処理時間が既知のサンプルプログラムを用意することで、制御プログラムの実行処理を見積もることができる制御プログラム開発支援装置またはプログラマブルコントローラを得ることを目的とする。

【００３４】

第４に、中間的に生成される高級プログラミング言語のソースファイルと実際の制御プログラムのソースファイルとの整合性を図りつつ、両ソースファイルに対するデバッギングが可能な制御プログラム開発支援装置またはプログラマブルコントローラを得ることを目的とする。

【００３５】

第５に、制御プログラムのうち変更箇所のみが記述された差分データに基づいて実行中の実行コードを新たな実行コードに自動置換するプログラマブルコントローラを得ることを目的とする。

【００３６】

第６に、制御プログラムのソースファイルを圧縮して実行コードに挿入することにより、制御プログラムと実行コードの一元化を図ることができる制御プログラム開発支援装置およびプログラマブルコントローラを得ることを目的とする。

【００３７】

第７に、命令解釈部において、制御プログラム中に多く使用されている命令を優先的にパターンマッチング対象とすることにより、コンパイルやインタプリーティング等の逐次命令解釈に要する時間の短縮化を図ることができる制御プログラム開発支援装置およびプログラマブルコントローラを得ることを目的とする。

【００３８】

【課題を解決するための手段】

上述した課題を解決し、目的を達成するため、この発明にかかるプログラマブルコントローラにあっては、制御プログラムをコンパイルすることで生成された実行コードに基づいてシーケンス処理をおこなうプログラマブルコントローラにおいて、前記実行コードを記憶する記憶手段と、パイプラインロジックやキャッシュ等の高速化実装手段を搭載するとともに前記実行コードによって直接実行される汎用マイクロプロセッサと、を備えたことを特徴とする。

【００３９】

この発明によれば、高速化実装手段を搭載した汎用マイクロプロセッサを備えるとともに、受信した実行コードが、その汎用マイクロプロセッサの高速化実装手段を用いて実行されるように最適化されているので、パーソナルコンピュータ等に使用される汎用マイクロプロセッサのパイプライン、スーパスカラ、アウトオブオーダ等の高速処理によってシーケンス動作を実行することができる。

【００４０】

つぎの発明にかかる制御プログラム開発支援装置にあっては、ラダーダイヤグ

ラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、前記制御プログラムを、パイプラインロジックやキャッシュ等の高速化実装手段を搭載した汎用マイクロプロセッサにおいて直接実行可能な実行コードにコンパイルするコンパイル手段を備えたことを特徴とする。

【００４１】

この発明によれば、シーケンス制御用の言語で記述された制御プログラムを、高速化実装手段を搭載した汎用マイクロプロセッサで実行できる実行コードにコンパイルするので、そのような汎用マイクロプロセッサを具備したプログラマブルコントローラにその実行コードを送信することで、パイプライン、スーパスカラ、アウトオブオーダ等の高速処理によるシーケンス動作を実行させることができるとともに、コンパイル手段を変更することで、様々な汎用マイクロプロセッサに対応した実行コードを得ることができる。

【００４２】

つぎの発明にかかる制御プログラム開発支援装置にあっては、ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、前記制御プログラムを複数のブロックに分割する制御プログラム分割手段と、前記複数のブロックのすべてまたは一部をプログラマブルコントローラにおいて直接実行可能な実行コードにコンパイルするコンパイル手段と、前記実行コードを前記プログラマブルコントローラに送信する通信インターフェースと、を備えたことを特徴とする。

【００４３】

この発明によれば、制御プログラムを複数のブロックに分割して、分割したブロック毎にコンパイルして最終的な実行コードを得ることができるので、長大な一本の制御プログラムであっても、変更の生じた一部分だけを再コンパイルすることで実行コードを生成することができる。

【００４４】

つぎの発明にかかる制御プログラム開発支援装置にあっては、ラダーダイヤグ

ラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、前記制御プログラムを複数のブロックに分割する制御プログラム分割手段と、前記複数のブロックのすべてまたは一部を、当該ブロック毎に、汎用コンピュータ用高級言語で記述された高級言語制御プログラムに変換する制御プログラム変換手段と、前記ブロック毎に対応する汎用コンピュータ用高級言語のすべてまたは一部をプログラマブルコントローラにおいて直接実行可能な実行コードにコンパイルするコンパイル手段と、を備えたことを特徴とする。

【００４５】

この発明によれば、制御プログラムを複数のブロックに分割して、分割したブロックのそれぞれを汎用コンピュータ用高級言語に変換する。そして、各ブロックに対応する汎用コンピュータ用高級言語のプログラムをコンパイルして最終的な実行コードを得ることができるので、長大な一本の制御プログラムであっても、変更の生じた一部分だけを再コンパイルすることで実行コードを生成することができるとともに、コンパイラとして従来の汎用コンピュータ用高級言語用のコンパイラを使用することができる。

【００４６】

つぎの発明にかかる制御プログラム開発支援装置にあっては、上記発明において、前記プログラマブルコントローラが、パイプラインロジックやキャッシュ等の高速化実装手段を搭載する汎用マイクロプロセッサを備えたことを特徴とする。

【００４７】

この発明によれば、パーソナルコンピュータ等に使用される汎用マイクロプロセッサを搭載したプログラマブルコントローラに対して実行コードを生成するので、パイプライン、スーパスカラ、アウトオブオーダ等の高速処理によるシーケンス動作を実行させることができる。

【００４８】

つぎの発明にかかる制御プログラム開発支援装置にあっては、上記発明において、前記制御プログラムが、ラダーダイヤグラムまたはラダーダイヤグラムから

生成されたインストラクションリストであり、前記制御プログラム分割手段が、前記ラダーダイヤグラム内の所定のラングにおいて複数のブロックに分割して、当該ブロック毎にプログラムファイルを生成することを特徴とする。

【００４９】

この発明によれば、制御プログラム分割手段が、ラダーダイヤグラム内の所定のラングの切れ目において複数のブロックに分割し、当該ブロック毎にプログラムファイルを生成するので、入力から出力までの一連の回路ブロックを単位としてプログラムファイルを管理することができるとともに、コンパイル時における入出力デバイス値を保有するレジスタ変数の最適割り当てが分割によって阻害されるという事態を低減させることができる。特に、高速化実装手段を搭載したマイクロプロセッサに対しては、複数のラングの集まりで一つのブロックを構成することで、コンパイラのレジスタ利用効率の向上や最小演算割り当ての最適化機能を活用することができる。

【００５０】

つぎの発明にかかる制御プログラム開発支援装置にあっては、上記発明において、前記制御プログラムが、ラダーダイヤグラムまたはラダーダイヤグラムから生成されたインストラクションリストであり、前記制御プログラム分割手段が、前記ラダーダイヤグラム内のジャンプ命令に対するジャンプ先となる所定のラングにおいて複数のブロックに分割し、当該ブロック毎にプログラムファイルを生成することを特徴とする。

【００５１】

この発明によれば、制御プログラム分割手段が、ラダーダイヤグラム内の所定のラングの切れ目であってかつジャンプ命令に対するジャンプ先となる位置において複数のブロックに分割し、当該ブロック毎にプログラムファイルを生成するので、コンパイラの最適化に影響を与えないとともに、分割後のプログラムファイルをサブルーチンとして管理することができる。

【００５２】

つぎの発明にかかる制御プログラム開発支援装置にあっては、上記発明において、前記制御プログラムが、ラダーダイヤグラムまたはラダーダイヤグラムから

生成されたインストラクションリストであり、前記制御プログラム分割手段が、前記ラダーダイヤグラムにおいて共通の入力デバイスまたは出力デバイスに対する命令を含んだラングのすべてまたは一部を抽出し、抽出したラングのすべてまたは一部により一つのブロックを構成するとともに当該ブロック毎にプログラムファイルを生成することを特徴とする。

【００５３】

この発明によれば、制御プログラム分割手段が、ラダーダイヤグラムにおいて共通の入力デバイスまたは出力デバイスに対する命令を含んだラングのすべてまたは一部を抽出し、抽出したラングのすべてまたは一部を一つのブロックとして複数のブロックに分割し、当該ブロック毎にプログラムファイルを生成するので、特定のデバイスに対する制御プログラムの変更が生じた際に、その部分に該当するプログラムファイルのみを変更して再コンパイルして実行コードを得ることができる。

【００５４】

つぎの発明にかかる制御プログラム開発支援装置にあっては、上記発明において、前記制御プログラムを、参照されない変数の排除、冗長コードの排除、共通の入力デバイスや出力デバイスに対する命令を局所的に配置するためのコードの並べ替え等をおこなうことによって最適なコード体系に再構築する最適化フィルタ手段を備え、前記最適化フィルタ手段により最適化された制御プログラムをあらためて前記制御プログラムとすることを特徴とする。

【００５５】

この発明によれば、制御プログラムを、コンパイルする前段において、一旦、参照されない変数の排除、冗長コードの排除、並べ替え等をおこなうことによって最適なコード体系に再構築するので、最適化されたコードにより構成された制御プログラムをコンパイル手段に提供することができる。

【００５６】

つぎの発明にかかる制御プログラム開発支援装置にあっては、上記発明において、処理時間が既知であるサンプルプログラムと前記実行コードに対応する制御プログラムとを対応付けた対応表を有し、当該対応表に基づいてプログラマブル

コントローラにおけるシーケンス処理実行時間を見積もる処理時間概算手段を備えたことを特徴とする。

【００５７】

この発明によれば、処理時間が既知であるサンプルプログラムと実行コードに対応する制御プログラムとを対応付けた対応表から、シーケンス処理実行時間を見積もることができるので、従来困難であった制御プログラム内の各インストラクションの処理時間を知得することが可能となる。

【００５８】

つぎの発明にかかる制御プログラム開発支援装置にあっては、ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、前記制御プログラムを汎用コンピュータ用高級言語で記述された高級言語制御プログラムに変換する制御プログラム変換手段と、前記高級言語制御プログラムを構成するソースコード内において、前記インストラクションリストを構成する各行に対応した部分に当該行番号を示すコードを挿入してデバッグ用制御プログラムを生成するデバッグコード生成手段と、前記デバッグ用制御プログラムをステップ実行することにより前記インストラクションリストの各行と前記高級言語制御プログラムの実行部分とを対応付けて表示するデバッグ実行手段と、を備えたことを特徴とする。

【００５９】

この発明によれば、インストラクションリストとそのインストラクションリストを変換することで得られた高級言語制御プログラムとの実行部分を対応付けてステップ実行できるとともにその部分を表示することができるので、インストラクションリストをコンパイルして得られた実行コードに基づいて動作するプログラマブルコントローラにおいても、ユーザが記述したインストラクションリストのステップ実行レベルで動作確認がおこなえるようになる。

【００６０】

つぎの発明にかかるプログラマブルコントローラにあっては、制御プログラムをコンパイルすることで生成された実行コードに基づいてシーケンス処理をおこ

なうプログラマブルコントローラにおいて、前記実行コードを記憶する第１の記憶手段と、前記第１の記憶手段に記憶されている実行コードと新たな実行コードとの差分データを記憶する第２の記憶手段と、前記実行コードにより直接実行されるマイクロプロセッサと、前記差分データに基づいて、所定のタイミングで、現在実行中の実行コードを新たな実行コードに変更するとともに、変更後の実行コードを継続的に実行させるパッチ処理手段と、を備えたことを特徴とする。

【００６１】

この発明によれば、現在実行されている実行コードと新たな実行コードとの差分データに基づいて、現在実行中の実行コードを、所定のタイミングで新たな実行コードに変更し、変更後の実行コードを継続的に実行させるので、実行コードに変更が生じた場合であっても、実行コードのうち、変更の生じたバイナリコードのみを修正して新たな実行コードに変更することができるとともに、プログラマブルコントローラの実行状態を中断させることなく、新たなシーケンス動作を継続させることができる。

【００６２】

つぎの発明にかかるプログラマブルコントローラにあっては、制御プログラムをコンパイルすることで生成された実行コードに基づいてシーケンス処理をおこなうプログラマブルコントローラにおいて、前記実行コードを記憶する記憶手段と、前記実行コードにより直接実行されるマイクロプロセッサと、を備え、前記実行コードは、前記制御プログラムの圧縮により生成されたバイナリデータを含むことを特徴とする。

【００６３】

この発明によれば、制御プログラムと実行コードとを一元化してプログラマブルコントローラに保持するので、開発やデバッグ時に必要となる制御プログラムを消失した場合や現場のプログラマブルコントローラから直接制御プログラムを操作したい場合において、実行コードからその制御プログラムを取り出すことができる。

【００６４】

つぎの発明にかかる制御プログラム開発支援装置にあっては、ラダーダイヤグ

ラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、前記制御プログラムを圧縮して圧縮ファイルを生成する圧縮手段と、前記圧縮ファイルを前記制御プログラムのコード体系に変換した圧縮データを生成するコード変換手段と、前記制御プログラムと前記圧縮データとを結合するとともに、当該結合結果を、プログラマブルコントローラにおいて直接実行可能な実行コードにコンパイルするコンパイル手段と、を備えたことを特徴とする。

【００６５】

この発明によれば、制御プログラムと実際に実行される実行コードとを一元化してあらためて実行コードとして生成するので、開発やデバッグ時に必要となる制御プログラムを消失した場合や現場のプログラマブルコントローラから直接制御プログラムを操作したい場合において、実行コードからその制御プログラムを取り出すことができる。

【００６６】

つぎの発明にかかるプログラマブルコントローラにあっては、ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムに基づいて、シーケンス処理をおこなうプログラマブルコントローラにおいて、前記制御プログラムを記憶する記憶手段と、前記制御プログラムに使用されている各命令の出現頻度をカウントする命令カウント手段と、前記命令カウント手段によってカウントされた結果から、出現頻度の高い順に命令を列挙したパターンマッチング表を作成するパターンマッチング表作成手段と、前記制御プログラムを、前記パターンマッチング表に列挙されている命令順にパターンマッチングしつつ、前記プログラマブルコントローラにおいて直接実行可能な実行コードにインタプリーティングしつつ実行するインタプリーティング手段と、を備えたことを特徴とする。

【００６７】

この発明によれば、制御プログラムを実行コードにインタプリーティングする際に、制御プログラムのうち出現頻度の高い命令順にパターンマッチングをおこなうので、インタプリーティングの際に、制御プログラムの各命令に適合する実

行コードを迅速に発見することができる。

【００６８】

つぎの発明にかかる制御プログラム開発支援装置にあっては、ラダーダイヤグラムやインストラクションリスト等のシーケンス制御用の言語で記述された制御プログラムを開発するための制御プログラム開発支援装置において、前記制御プログラムに使用されている各命令の出現頻度をカウントする命令カウント手段と、前記命令カウント手段によってカウントされた結果から、出現頻度の高い順に命令を列挙したパターンマッチング表を作成するパターンマッチング表作成手段と、前記制御プログラムを、前記パターンマッチング表に列挙されている命令順にパターンマッチングしつつ、前記プログラマブルコントローラにおいて直接実行可能な実行コードにコンパイルするコンパイル手段と、を備えたことを特徴とする。

【００６９】

この発明によれば、制御プログラムを実行コードにコンパイルする際に、制御プログラムのうち出現頻度の高い命令順にパターンマッチングをおこなうので、コンパイルの際に、制御プログラムの各命令に適合する実行コードを迅速に発見することができる。

【００７０】

つぎの発明にかかるプログラマブルコントローラにあっては、上記制御プログラム開発支援装置を備えたことを特徴とする。

【００７１】

この発明によれば、制御プログラムの開発とともにシーケンス処理も可能とする、いわゆるソフトウェアプログラマブルコントローラにおいて、上記した制御プログラム開発支援装置と上記したプログラマブルコントローラの各機能を実現することができる。

【００７２】

【発明の実施の形態】

以下に、この発明にかかるプログラマブルコントローラおよび制御プログラム開発支援装置の実施の形態を図面に基づいて詳細に説明する。なお、この実施の

形態によりこの発明が限定されるものではない。

【００７３】

実施の形態１．

まず、実施の形態１にかかるプログラマブルコントローラおよび制御プログラム開発支援装置について説明する。実施の形態１にかかるプログラマブルコントローラは、高速化実装技術により構築された汎用マイクロプロセッサを搭載したことを特徴としており、実施の形態１にかかる制御プログラム開発支援装置は、インストラクションリスト等のシーケンス処理用の言語で記述された制御プログラムをコンパイルして、上記汎用マイクロプロセッサにおいて直接実行可能な実行コードを生成することを特徴としている。

【００７４】

図１は、実施の形態１にかかるプログラマブルコントローラの概略構成を示すブロック図である。図１において、プログラマブルコントローラ１０は、実行コードに従ったシーケンス処理を実行する汎用マイクロプロセッサ１１と、実行コード等を記憶するメモリ１４と、実行コードであるバイナリファイル等を制御プログラム開発支援装置２０から受信するとともに必要に応じて上記メモリ１４内のプログラム等を外部に送信する通信インターフェース１６と、汎用マイクロプロセッサ１１による指示に応じて、外部の入力デバイス２１から信号を入力するとともに外部の出力デバイスに制御信号を出力する入出力制御部１５と、を備えて構成される。

【００７５】

特に、汎用マイクロプロセッサ１１は、図１に示しているように、ストアードプログラム方式等の基本的な演算処理をおこなうＣＰＵコア１２と、高速化実装技術部１３と、を備えており、高速化実装技術部１３では、命令キャッシュおよびデータキャッシュを備えて構築され、２次キャッシュ、パイプラインロジックおよびスケーラロジックを備えている。すなわち、この汎用マイクロプロセッサ１１は、パイプライン処理やスーパースケーラ処理の他、アウトオブオーダ、分岐予測、投機実行等の高速化処理が可能なマイクロプロセッサである。なお、図示した汎用マイクロプロセッサ１１においては、高速化実装技術部１３を、ＣＰ

Ｕコア１２と分離して記載したが、ＣＰＵコア１２中に、高速化実装技術部１３中の各機能ブロックが複数個保有した形態も存在する。

【００７６】

また、メモリ１４は、汎用マイクロプロセッサ１１が演算処理に使用するとともに、実行コードうちの実際の実行部分またはすべてをロードさせておくためのワークメモリや、ＢＩＯＳ（Ｂａｓｉｃ　Ｉｎｐｕｔ　Ｏｕｔｐｕｔ　Ｓｙｓｔｅｍ）が記録されたＲＯＭ、実行コードのすべてを保持しておくフラッシュメモリ等の不揮発性メモリ等から構成される。なお、図示しないが、プログラマブルコントローラ１０は、実行コード、その他プログラムまたはデータを記憶しておくための磁気ディスク装置やメモリカード等の二次記憶手段を備えることもできる。

【００７７】

すなわち、図１に示すプログラマブルコントローラ１０は、従来のＡＳＩＣにより構成されたマイクロプロセッサを、高速化実装手段を備えた汎用マイクロプロセッサに置換した点以外は、従来のプログラマブルコントローラの構成を適用することができる。

【００７８】

よって、このプログラマブルコントローラ１０にシーケンス処理を実行させるには、上記汎用マイクロプロセッサ１１において直接実行可能な実行コードが必要となるが、この実行コードは、制御プログラム開発支援装置２０によって生成される。

【００７９】

制御プログラム開発支援装置２０は、プログラマブルコントローラ１０と通信可能なインターフェースを備えていれば、パーソナルコンピュータ等の汎用のコンピュータシステムと同構成により実現することができる。

【００８０】

特に、この制御プログラム開発支援装置２０は、ラダーダイヤグラムやインストラクションリストによって記述された制御プログラムのファイルを読み込み、読み込んだファイルのコード体系を、上記汎用マイクロプロセッサ１１において

直接実行可能な実行コードにコンパイルするコンパイラを備えている。図２は、実施の形態１にかかる制御プログラム開発支援装置において、実行コードの生成を説明するための説明図である。

【００８１】

図２に示すように、制御プログラム開発支援装置２０は、既に開発済みまた新規に開発されたラダーダイヤグラムやインストラクションリスト等のシーケンス処理用の言語で記述された制御プログラム１を、コンパイラ１００に投入することで実行コード９を生成する。コンパイラ１００は、ソフトウェアプログラムにより実現してもハードウェアロジックにより実現してもよいが、上記汎用マイクロプロセッサ最適化技術を利用することで、汎用マイクロプロセッサ１１の機能を最大限に発揮する実行コードを生成する必要がある。

【００８２】

以上に説明したとおり、実施の形態１にかかるプログラマブルコントローラによれば、ＡＳＩＣとして専用のマイクロプロセッサの設計および開発をおこなうことなく、比較的安価で入手しやすい汎用マイクロプロセッサ１１を用いてプログラマブルコントローラを構成することができるとともに、汎用マイクロプロセッサ１１の高速化実装技術によってシーケンス処理をより高速に実行することが可能になる。

【００８３】

また、実施の形態１にかかる制御プログラム開発支援装置によれば、シーケンス処理用の言語で記述された制御プログラムをコンパイルすることにより、上記した汎用マイクロプロセッサ１１において直接実行可能な実行コード９を得ることができるので、汎用マイクロプロセッサ１１を搭載したプログラマブルコントローラに対して、実行コード９を転送してシーケンス処理を実行させることができ、過去の制御プログラムの資産を有効に活用することができる。さらに、搭載する汎用マイクロプロセッサ１１の種類が異なるプログラマブルコントローラ間に対しても、使用するコンパイラ１００を入れ替えることで、実行コード９を得ることができるため、プログラマブルコントローラのメーカごとまたは機種ごとに制御プログラムを開発する必要がなくなり、同内容のシーケンス処理を記述し

た制御プログラムを一元的に管理することができるとともに、開発時間やデバッギング時間の低減を図ることができる。

【００８４】

実施の形態２．

つぎに、実施の形態２にかかる制御プログラム開発支援装置について説明する。実施の形態２にかかる制御プログラム開発支援装置は、インストラクションリスト等のシーケンス処理用の言語で記述された制御プログラムを、所定位置で複数のプログラムファイルに分割し、分割したプログラムファイルの個々のみを修正してコンパイルすることができることを特徴としている。

【００８５】

図３は、実施の形態２にかかる制御プログラム開発支援装置において、制御プログラムの分割をともなう実行コード生成処理を説明するための説明図である。図３に示すように、制御プログラム開発支援装置は、後述する分割アルゴリズムに基づいて、既に開発済みまた新規に開発されたラダーダイヤグラムやインストラクションリスト等のシーケンス処理用の言語で記述され、かつ従来において一つのプログラムファイルとして取り扱われていた制御プログラム１を、複数の分割プログラム３－１～３－Ｎに分割する。

【００８６】

つぎに、分割アルゴリズムについて説明する。制御プログラムを分割する目的は、制御プログラムに変更が生じた際の再コンパイルに要する時間を短縮することであるが、コンパイラ１００がおこなう最適化の処理に対して、大きな妨げとなるような分割をおこなうべきではない。制御プログラムを分割すると、全く分割をおこなっていない制御プログラムをコンパイルする場合と比べて、完全に最適な実行コードを得ることはできないため、分割に際しては、完全に最適な実行コードに近いコード、すなわち最適化のレベルをできるだけ落とさないコードを出力するような工夫が必要である。

【００８７】

そこで、分割アルゴリズムとしては、（１）指定ステップごとのラングの切れ目でおこなう方法（以下、ラング分割方法と称する。）、（２）ジャンプ先であ

るサブルーチンでおこなう方法（以下、ジャンプ先分割方法と称する。）、（３）使用されている入力デバイスまたは出力デバイスを機器ごとにひとまとめにする方法（以下、機器分割方法と称する。）の３通りが考えられる。

【００８８】

特に、これら３通りの分割アルゴリズムは、いずれもラダーダイヤグラムの表現を基礎としているので、ここではまず、ラダーダイヤグラムについて説明する。図４は、ラダーダイヤグラムの例を示す図である。図４において、インストラクションＣＭ１～ＣＭ６は、シーケンス演算で使用するコマンドに対応する。特に、図中、インストラクションＣＭ１、ＣＭ２およびＣＭ５は入力を意味し、ＣＭ３およびＣＭ６は出力を意味し、ＣＭ４は演算処理を意味する。また、ラダーダイヤグラムでは、入力から出力までのひとまとまりを一つの回路ブロックに分けて表現され、これはラングと呼ばれている。

【００８９】

図４においては、ラングＲ１とＲ２の二つのラングが示されている。さらに、各ラングには、インストラクションの処理時間を特定の単位で示したステップ数が付記される。図４中のラングＲ１では、入力ポートに割り当てられたアドレスへのデータの読み込みに１ステップ、出力ポートに割り当てられたアドレスへのデータの書き出しに１ステップ、アドレス間においてデータを移動させるＭＯＶ命令に２ステップかかるため、つぎのラングＲ２の先頭のステップ数は５となっている。

【００９０】

すなわち、ラングＲ１では、メモリＸ１およびＸ２を入力デバイスとしており、メモリＹ１を出力デバイスとしている。また、ラングＲ２では、メモリＸ１を入力デバイスとしており、メモリＹ１０を出力デバイスとしている。

【００９１】

（１）ラング分割方法

まず、ラング分割方法について説明する。ラング分割方法は、制御プログラムの分割を、上記したラダーダイヤグラムによって表されるラングの切れ目でおこなう方法である。ラングの切れ目で分割することで、コンパイラによるデバイス

のレジスタへの配置を有効に活用し、分割による最適化の効果低減を抑えることができる。

【００９２】

シーケンス処理はラングの集まりであると考えることができ、入力デバイスの情報に対して与えられたシーケンス演算をおこない、出力デバイスの値を求めることが、ラングの処理である。よって、シーケンス処理の高速化を図るには、各ラングでの処理を速くする必要がある。また、一つのラングの途中で分割すると、レジスタに配置されているラングの途中までの演算結果を有効に活用することができないことからも、制御プログラムの分割をラングの切れ目でおこなうことが好ましい。

【００９３】

そこで、コンパイラ１００を用いて入力デバイスおよび出力デバイスをできるだけレジスタへ配置することで、高速な演算を実現する。すなわち、コンパイラの最適化機能を利用し、レジスタを積極的に活用する。

【００９４】

しかしながら、制御プログラム１をラングの切れ目ごとに細切れにすると、実施の形態１において説明したような汎用マイクロプロセッサの実行コードを生成しようとする場合には、コンパイラ１００において、高速化実装技術の一つである最適命令スケジューリングにより、アウトオブオーダ化されたコードを出力するための最適化機能を使用しないことになり、最適化の度合いが減少する。

【００９５】

よって、分割後の制御プログラム、すなわち分割プログラム３－１～３－Ｎは、ある程度のサイズを持っていることが望ましい。そこで、分割する位置を特定のステップ数単位で指定する。例えば、１００ステップの倍数毎にラングの切れ目で制御プログラムを分割する。１００ステップの倍数のステップの位置がラングの切れ目ではない場合は、その位置の直後のラングの切れ目で分割をおこなう。

【００９６】

このようにして生成された分割プログラム３－１～３－Ｎは、それら分割プロ

グラムごとに独立してコンパイルすることができるように、それぞれ個別のプログラムファイルとして管理される。図５は、ステップ数と分割後ファイルの対応関係を示した表である。図５に示すように、分割プログラムにはそれぞれ分割後ファイル名が付され、さらに、各ファイルには、元の制御プログラム１のどのラングに相当するかを示したステップ数が割り当てられる。

【００９７】

制御プログラムに修正の必要が生じた場合には、分割前の制御プログラム１に対して、例えばラダーダイヤグラム上での修正作業をおこなうが、上記表は、制御プログラム１において実際に修正された部分のステップ数を抽出し、その修正された部分を含む分割プログラムのファイル名を特定する際に活用される。

【００９８】

つぎに、ラング分割方法の具体的な処理の流れについて説明する。図６は、ラング分割方法の処理を示したフローチャートである。図６において、まず、ラング分割方法を実行するコンピュータプログラム等の分割手段が、制御プログラム１であるインストラクションリスト（以下、ＩＬと称する。）、特にラダーダイヤグラムを変換して得られたＩＬファイルを、１行読み込む（ステップＳ１０１）。読み込んだ１行が、ＩＬファイルの終わりであれば処理を終了する（ステップＳ１０２肯定）。

【００９９】

一方、読み込んだ１行が、ＩＬファイルの終わりではなく（ステップＳ１０２否定）、指定したステップ数以上に相当し、かつラングの切れ目であれば（ステップＳ１０３肯定）、新規にファイルを作成して、そのファイルをカレントファイルに設定する（ステップＳ１０４）。なお、ラングの切れ目であるか否かは、インストラクションの種類によって判断する。すなわち、出力をおこなうインストラクションの直後に入力をおこなうインストラクションが現れたところがラングの切れ目となる。また、新規に作成するファイルは、ファイル名を連番にするなど、一意の名前になるようにする。

【０１００】

そして、読み出した１行を、既に設定されているカレントファイルに書き込み

（ステップＳ１０５）、再度ステップＳ１０１からの処理を繰り返す。すなわち、読み込んだ１行がラングの切れ目でなければ（ステップＳ１０３否定）、または、指定したステップ数に達していなければ、つぎつぎとその１行をカレントファイルに書き込んでいく。これにより、ラングの切れ目であってかつ所定ステップ数の範囲ごとに、ファイルが作成される。このファイルが上記した分割プログラムごとに対応して生成される分割後ファイルである。

【０１０１】

（２）ジャンプ先分割方法

つぎに、ジャンプ先分割方法について説明する。ジャンプ先分割方法は、制御プログラムの分割を、上記したラダーダイヤグラムによって表されるラングの切れ目であって、かつジャンプ命令のジャンプ先となるサブルーチンでおこなう方法である。これにより、ジャンプ命令に影響のある最適化に対して配慮することができる。

【０１０２】

図７は、ジャンプ先ラベル名と分割後ファイルの対応関係を示した表である。図７に示すように、分割プログラムにはそれぞれ分割後ファイル名が付され、さらに、各ファイルには、ジャンプ命令のジャンプ先を示すラベル名が割り当てられる。

【０１０３】

図７に示す表も、図５に示した表と同様に、制御プログラム１において実際に修正された部分を含むサブルーチンのラベル名を抽出し、そのサブルーチンに相当する分割プログラムのファイル名を特定する際に活用される。

【０１０４】

つぎに、ジャンプ先分割方法の具体的な処理の流れについて説明する。図８は、ジャンプ先分割方法の処理を示したフローチャートである。図８において、まず、ジャンプ先分割方法を実行するコンピュータプログラム等の分割手段が、制御プログラム１であるＩＬ、特にラダーダイヤグラムを変換して得られたＩＬファイルを、１行読み込む（ステップＳ２０１）。読み込んだ１行が、ＩＬファイルの終わりであれば処理を終了する（ステップＳ２０２肯定）。

【０１０５】

一方、読み込んだ１行が、ＩＬファイルの終わりではなく（ステップＳ２０２否定）、ラングの切れ目であり、かつジャンプ先ラベルであれば（ステップＳ２０３肯定）、新規にファイルを作成して、そのファイルをカレントファイルに設定する（ステップＳ２０４）。なお、ラングの切れ目であるか否かの判断およびファイル名の決定方法は、上述したラング分割方法と同様である。

【０１０６】

そして、読み出した１行を、既に設定されているカレントファイルに書き込み（ステップＳ２０５）、再度ステップＳ２０１からの処理を繰り返す。すなわち、読み込んだ１行がラングの切れ目でなくかつジャンプ先のサブルーチンの先頭でなければ、つぎつぎとその１行をカレントファイルに書き込んでいく。これにより、ジャンプ先のラベル名が付されている間ごとに、ファイルが作成される。このファイルが上記した分割プログラムごとに対応して生成される分割後ファイルである。

【０１０７】

（３）機器分割方法

つぎに、機器分割方法について説明する。シーケンス処理では、一般に、製造ライン等を構成する複数の機器ごとに、使用している入力デバイスおよび出力デバイスは異なる。製造現場において生じる制御プログラムの変更は、複数の機器のうち、ある機器に接続された入力デバイスを変更するなど機器に依存することが多いという特性を有している。そこで、機器分割方法では、機器ごとに対応するラングをひとまとまりにしてファイルに出力する。

【０１０８】

つぎに、機器分割方法の具体的な処理の流れについて説明する。図９は、機器分割方法の処理を示したフローチャートである。図９において、まず、上述したラング分割方法と同様に、ラングの切れ目であって、かつ同一の機器におけるシーケンス処理を示すラングを抽出して分類する。そして、機器ごとに分類された複数のラングをひとまとまりとして、それぞれ各機器に対応するファイルを作成する（ステップＳ３０１）。

【０１０９】

つづいて、これら複数のファイル内においてまたがって使用されている入力デバイスおよび出力デバイスの数を算出する（ステップＳ３０２）。算出されたデバイス数が指定数以下であれば、ステップＳ３０１において作成された複数のファイルを、制御プログラム１を複数の分割プログラム３－１～３－Ｎに分割した結果とする（ステップＳ３０３肯定）。一方、算出されたデバイス数が指定数より大きければ、分割の基準となるラング数の範囲を小さくするなどして、再度、ステップＳ３０１に戻り、機器ごとの分類を繰り返す（ステップＳ３０３否定）。

【０１１０】

センサ情報を扱うデバイスなど、デバイスによっては、複数の機器で使用されていることが考えられるため、完全にデバイスが重ならないようなラングで分割することは困難である。しかし、上述したフローチャートに示す処理のように、分割後に分割したファイル間にまたがるデバイスの数を抑制することで、ラング分割方法において説明したような最適化の効果の低減を抑えることができる。なお、上記した機器ごとのラングの分類は、ユーザがおこなってもよい。

【０１１１】

よって、実施の形態２にかかる制御プログラム開発支援装置では、以上に説明した３通りの分割アルゴリズムのいずれかを採用することができる。このような分割アルゴリズムによって生成された複数の分割プログラムのファイルは、制御プログラム１に修正が生じた際に、効果的に活用される。

【０１１２】

図１０は、制御プログラム１に修正が生じた際の実行コード生成処理を説明するための説明図である。図１０に示すように、例えば、制御プログラム１の修正により、変更箇所２が生じた場合には、その変更箇所２を含む分割プログラム３－１に対応するファイルのみが、コンパイラ１００によって再コンパイルされて、その部分のみのオブジェクトファイルを得る。なお、このオブジェクトファイルは、修正前の分割プログラム３－１に対応するオブジェクトファイルと置換される。

【０１１３】

そして、リンカによって、この新たに生成されたオブジェクトファイルと、他の既に生成済みのオブジェクトファイル、すなわち変更の生じていない分割プログラムに対応するオブジェクトファイルと、がリンクされることで、新たな実行コード９が生成される。

【０１１４】

以上に説明したとおり、実施の形態２にかかる制御プログラム開発支援装置によれば、インストラクションリスト等のシーケンス処理用の言語で記述された制御プログラム１を、コンパイルに影響がないような位置で複数のプログラムに分割し、それら分割プログラムごとにコンパイルして実行コードを得ることができるので、制御プログラム１に修正が生じた場合であっても、その修正により変更された部分に該当する分割プログラムのみを再度コンパイルするため、そのような修正が生じた場合のコンパイル時間を短縮することができる。特に、プログラマブルコントローラが設置された現場においては、制御プログラム１の修正を要する箇所は特定の部分に集中するため、現場に、ハンディプログラムローダ等の制御プログラム開発支援装置を持ち込んで保守作業をおこなう場合に、その作業時間を短縮させることが可能になる。

【０１１５】

なお、上記したコンパイラ１００は、実施の形態１において説明したように、高速化実装技術により構築された汎用マイクロプロセッサの実行コードを生成するものであってもよい。

【０１１６】

実施の形態３．

つぎに、実施の形態３にかかる制御プログラム開発支援装置について説明する。実施の形態３にかかる制御プログラム開発支援装置は、実施の形態２において生成された複数の分割プログラムを、コンバータによってそれぞれ高級プログラミング言語で記述されたファイルに変換し、それら高級プログラミング言語のファイルをコンパイルすることで実行コードを得ることを特徴としている。

【０１１７】

図１１は、制御プログラムの分割と高級プログラミング言語への変換とをともなう実行コード生成処理を説明するための説明図である。図１１に示すように、制御プログラム１を複数の分割プログラム３－１～３－Ｎに分割するまでの処理は、実施の形態２と同様である。

【０１１８】

実施の形態３にかかる制御プログラム開発支援装置では、それら複数の分割の分割プログラム３－１～３－Ｎを、コンバータ１０１に投入し、それぞれコンバータ１０１によって、複数の高級プログラミング言語のファイルに変換する点が実施の形態２と異なる。特に、図１１では、高級プログラミング言語としてＣ言語を採用した場合を示しており、分割プログラム３－１～３－Ｎは、それぞれＣ言語表現のファイル４－１～４－Ｎに変換される。

【０１１９】

コンバータ１０１によって生成された複数のＣ言語表現のファイル４－１～４－Ｎは、それぞれ独立したプログラムファイルとして取り扱われ、Ｃコンパイラ１０２に投入される。Ｃコンパイラ１０２は、これら複数のＣ言語表現のプログラムファイルを個々にコンパイルしてオブジェクトファイルを生成し、これにより実行コード９が得られる。なお、実行コード９は、実際には、Ｃコンパイラ１０２によって生成された複数のオブジェクトファイルを、図示しないリンカによってリンクされることにより生成される。

【０１２０】

図１２は、高級プログラミング言語に一旦変換する場合において、制御プログラム１に修正が生じた際の実行コード生成処理を説明するための説明図である。図１２に示すように、例えば、制御プログラム１の変更箇所２に相当する部分の修正が必要な場合には、その変更箇所２を含む分割プログラム３－１に対応するＣ言語表現のプログラムファイル４－１を直接操作して変更をおこなうことができる。なお、修正の範囲が大きい場合には、制御プログラム１を直接変更して、再度分割プログラムを生成してもよい。

【０１２１】

修正作業が終わると、Ｃコンパイラ１０２によって、その変更のあったＣ言語

表現のプログラムファイル４－１のみを再コンパイルし、分割プログラム３－１に対応するオブジェクトファイルを得る。なお、このオブジェクトファイルは、修正前の分割プログラム３－１に対応するオブジェクトファイルと置換される。

【０１２２】

そして、リンカによって、この新たに生成されたオブジェクトファイルと、他の既に生成済みのオブジェクトファイル、すなわち変更の生じていない分割プログラムに対応するオブジェクトファイルと、がリンクされることで、新たな実行コード９が生成される。

【０１２３】

図１１および図１２に示した例では、制御プログラム１が、コンバータ１０１の前段において既に複数の分割プログラム３－１～３－Ｎに分割されるものとしていたが、複数の分割プログラム３－１～３－Ｎの生成をおこなわずに、制御プログラム１をＣ言語等の高級プログラミング言語に変換する際に分割をおこなってもよい。

【０１２４】

図１３は、コンバータが制御プログラム１の分割と高級プログラミング言語への変換をおこなう場合の実行コード生成処理を説明するための説明図である。図１３に示すように、分割プログラムは生成されずに、コンバータ１０３が、制御プログラム１を複数のＣ言語表現のプログラムファイル４－１～４－Ｎに分割する。これらプログラムファイル４－１～４－Ｎは、図１１に示したものと同様の内容である。すなわち、コンバータ１０３の内部処理としては、上述した分割アルゴリズムに従って、一旦、インストラクションリストのコード体系で複数の分割プログラムに分割し、それぞれの分割プログラムに対して高級プログラミング言語の変換をおこなう。

【０１２５】

図１４は、コンバータが制御プログラム１の分割と高級プログラミング言語への変換をおこなう場合において、制御プログラム１に修正が生じた際の実行コード生成処理を説明するための説明図である。図１４に示すように、制御プログラム１の変更箇所２に相当する部分の修正が必要な場合には、その変更箇所２を含

む分割プログラムに対応するＣ言語表現のプログラムファイル４－１を直接操作して変更をおこなうことができる。なお、修正の範囲が大きい場合には、制御プログラム１を直接変更して、再度分割されたＣ言語表現のプログラムファイルを生成してもよい。

【０１２６】

修正作業が終わると、Ｃコンパイラ１０２によって、その変更のあったＣ言語表現のプログラムファイル４－１のみを再コンパイルし、オブジェクトファイルを得る。新たな実行コード９の生成については、図１２の説明とおりである。

【０１２７】

以上に説明したとおり、実施の形態３にかかる制御プログラム開発支援装置によれば、インストラクションリスト等のシーケンス処理用の言語で記述された制御プログラム１を、コンパイルに影響がないような位置で複数のプログラムに分割し、それら分割プログラムを、さらにＣ言語等の高級プログラミング言語に変換して実行コードを得るので、実施の形態２において得られる効果を享受することができるとともに、修正が生じた場合、高級プログラミング言語で記述されたプログラムファイルに変更を加えることで目的の実行コードを得ることができ、制御プログラムの開発やデバッギングに要する時間をさらに短縮することができる。

【０１２８】

実施の形態４．

つぎに、実施の形態４にかかる制御プログラム開発支援装置について説明する。実施の形態４にかかる制御プログラム開発支援装置は、インストラクションリスト等の制御プログラムを、実施の形態１～３において説明したコンパイラやコンバータに入力する前に、一旦、最適化フィルタによってコードの再構成をおこなうことで、コンパイラにおいて、よりコンパクトで最適な実行コードが得られることを特徴としている。

【０１２９】

図１５は、実施の形態４にかかる制御プログラム開発支援装置の実行コード生成処理を説明するための説明図である。図１５に示すように、制御プログラム１

を、コンパイラ１０５に投入する前段において、最適化フィルタ１０４により最適化処理を施す。最適化フィルタ１０４では、参照されない変数の排除、冗長コードの排除、ロジックの組替え、共通の入力デバイスや出力デバイスに対する命令を局所的に配置するための順序の入れ替え等により、元の制御プログラム１のシーケンス処理内容を変えることなく最適化をおこない、最適化された制御プログラム５を出力する。特に、最適化処理にあたって、制御プログラム１中でアクセスするデバイスや使用する命令を近くに集めることで、プログラムに局所性を持たせることができ、コンパイル後の実行コードの実行効率を向上させることができる。

【０１３０】

最適化された制御プログラム５から実行コード９を生成するまでの処理の流れについては、実施の形態１～３に示した実行コード生成処理を適用することができるので、ここではその説明を省略する。

【０１３１】

以上に説明したとおり、実施の形態４にかかる制御プログラム開発支援装置によれば、インストラクションリスト等のシーケンス処理用の言語で記述された制御プログラム１に対して、コンパイラ１０５に投入する前段において、コード体系の再構成をおこなうので、後段のコンパイラ１０５において、よりコンパクトで最適な実行コード９を生成することができる。

【０１３２】

実施の形態５．

つぎに、実施の形態５にかかる制御プログラム開発支援装置について説明する。実施の形態５にかかる制御プログラム開発支援装置は、実行処理時間が既知であるサンプルプログラムを用意し、コンパイル前の制御プログラムの開発において、インストラクション等の実際の実行処理時間を見積もることを特徴としている。

【０１３３】

図１６は、実施の形態５にかかる制御プログラム開発支援装置において、インストラクションの例、すなわちサンプルプログラムとそのサイズとその処理時間

とを表した対応表である。図１６に示すように、様々なインストラクションを持ち、様々なサイズの制御プログラムのサンプルとそれらの実行処理時間を用意しておく。一般に、コンパイルされることにより生成された実行コードは、処理時間の見積もりが困難であるため、用意したサンプルプログラムの中から最も近いプログラムを選択し、作成した制御プログラムの処理時間の目安とする。なお、図１６に示した対応表を用いた処理時間の見積もりは、例えば、開発した制御プログラムにおいて、図１６に示すインストラクションが、所定の割合だけ含む部分を、そのインストラクションに近似する部分であると判断することで、コンピュータプログラム等により自動的におこなうこともできる。

【０１３４】

以上に説明したとおり、実施の形態５にかかる制御プログラム開発支援装置によれば、インストラクションリスト等の制御プログラム１を開発またはデバッギングする際に、実際の実行処理時間を見積もることができるので、シーケンス処理全体の処理速度やタイムテーブルの取得が容易となる。

【０１３５】

実施の形態６．

つぎに、実施の形態６にかかる制御プログラム開発支援装置について説明する。実施の形態６にかかる制御プログラム開発支援装置は、上述した実施の形態３において説明したように、インストラクションリスト等のシーケンス処理用の言語で記述された制御プログラムを、一旦、Ｃ言語等の高級プログラミング言語に変換する処理体系において、デバッギング作業時に、高級プログラミング言語のデバッグ部分が元の制御プログラムのどの部分に対応するかを知得することができることを特徴としている。

【０１３６】

ここでは、まず、実行コードを得るための制御プログラム開発環境について説明する。図１７は、制御プログラムの開発環境および実行環境を説明するための説明図である。図１７に示すように、開発環境３０は、エディタ１０６、コンバータ１０１、デバッガ１１０、コンパイラ１０８およびリンカ１０７を備えて構成される。エディタ１０６は、ラダーダイヤグラムやインストラクションリスト

、それらを変換して得た高級プログラミング言語のソースリスト等の制御プログラム１を作成するツールである。

【０１３７】

コンバータ１０１、コンパイラ１０８およびリンカ１０７は、実施の形態１～５において説明したものと同様のツールであり、ここではそれらの説明を省略する。デバッガ１１０は、作成した制御プログラムをデバッグするツールである。また、実行環境４０は、開発環境３０において生成された実行コード９を実際に実行する部分であり、これは、コンパイル型のプログラマブルコントローラであっても、制御プログラム開発支援装置上で上記プログラマブルコントローラをシミュレートしたエミュレータであってもよい。

【０１３８】

実施の形態６にかかる制御プログラム開発支援装置は、特にデバッギング時に作用および効果を有するため、ここでは、上記デバッガ１１０について詳細に説明する。図１８は、デバッギング作業を説明するための説明図である。デバッガ１１０は、実行中の制御プログラム１内において記述される各デバイスをモニタするモニタリング機能、制御プログラムをステップ実行するステップ実行機能、サンプリングトレース機能、ステータスラッチ機能、変数（すなわち、デバイス）が指定した値になるとブレークするブレーク機能、制御プログラム中にブレークポイントを設定するブレークポイント設定機能を有している。

【０１３９】

なお、図中、メモリ４１は、実施の形態１において説明したプログラマブルコントローラのメモリ１４と等価であるが、一般に、制御プログラム開発支援装置では、デバッグ時における実行環境４０を上記したエミュレータで実現する場合が多く、その場合には、メモリ４１は、制御プログラム開発支援装置に搭載されたメモリとなる。

【０１４０】

上記したモニタリング機能とは、制御プログラム１の実行中にデバイスの値を参照し、数値またはグラフ等のグラフィカルな手段で表示する機能であり、実行中にデバイスの値を書き換えることも可能である。

【０１４１】

また、サンプリングトレース機能とは、スキャン毎に指定したデバイスの値を記録する機能である。図１９は、このサンプリングトレース機能を説明するための説明図である。図１９に示すように、サンプリングトレース機能では、指定デバイス（図中においては出力デバイスＹ０およびＹ１が示されている。）のタイミングチャートを作成することができる。なお、このサンプリングトレースにおいて、デバイスの値を記録する領域は、デバッガ１１０の中に確保することも可能であるし、実行環境中の使用していないメモリを使用することもできる。また、実行環境中にサンプリングトレース用のメモリを確保していてもよい。

【０１４２】

ステータスラッチ機能とは、実行中のある瞬間の全デバイスの値を記録する機能である。図２０は、このステータスラッチ機能を説明するための説明図である。これらデバイスの値を記録する機能について記録を取るタイミング（すなわち、トリガ）は、指定デバイスが指定値になったとき、またはユーザがデバッガ１１０から手動により入力することができる。トリガがかかると、メモリの内容をステータスラッチメモリ４２へコピーする。このステータスラッチメモリ４２に保存されたメモリの内容は、デバッガ１１０によりモニタすることができる。

【０１４３】

つぎに、本実施の形態の特徴となるデバッギング手法について説明する。図２１は、そのデバッギング手法を説明するための説明図である。ここでは、インストラクションリストで記述された入力ソースファイル、すなわち制御プログラム１を、コンバータ１０１によってＣ言語等の高級プログラミング言語で記述された高級言語変換ファイル４に変換し、コンパイラ１０８がその高級言語変換ファイル４をコンパイルすることによって実行コード９を生成するシステムを考える。

【０１４４】

まず、コンバータ１０１は、制御プログラム１を高級言語変換ファイル４に変換する際に、制御プログラム１を構成する各インストラクションリストの行番号の情報を、高級言語変換ファイル４内の対応する部分に挿入する。この処理は、

図示しているように、一般的なＣ言語の処理系が保有しているソースファイルの行番号を認識するコンパイラ命令“＃ｌｉｎｅ”によって実現することができる。

【０１４５】

このコンパイラ命令“＃ｌｉｎｅ”は、Ｃ言語のソースファイルの各行が、別のソースファイルの各行に対応するといった、コンパイラに対する特殊命令である。一般的なＣコンパイラ、例えば、Ｆｒｅｅ　Ｓｏｆｔｗａｒｅ　ＦｏｕｎｄａｔｉｏｎのＧｎｕ　Ｃ　Ｃｏｍｐｉｌｅｒや、Ｍｉｃｒｏｓｏｆｔ社のＶｉｓｕａｌ　Ｃ／Ｃ＋＋コンパイラ等も、この機能をサポートしており、その構文は、

＃ｌｉｎｅ　行番号“ソースファイル名”

である。

【０１４６】

このように、元のインストラクションリストソースの各行に対応したコンパイラ命令を埋め込み、それを上記したＧｎｕ　ＣやＶｉｓｕａｌ　Ｃのようなコンパイラ１０８でコンパイルすることにより、入力ソースファイルの行番号情報が埋め込まれた実行可能形式のオブジェクトファイル、すなわち実行コード９を得ることができる。

【０１４７】

これを、各コンパイラに対応したデバッガ、例えばＧｎｕ　ＣならばＧｎｕ　Ｄｅｂｕｇｇｅｒ（ｇｄｂ）、Ｖｉｓｕａｌ　Ｃならば、Ｍｉｃｒｏｓｏｆｔ社の統合開発環境ＶｉｓｕａｌＳｔｕｄｉｏ付属のデバッガ上において、ステップ実行させると、各デバッガ上で元々の制御プログラム１が読み込まれ、その制御プログラム１上の行単位でステップ実行させることが可能になる。

【０１４８】

以上に説明したとおり、実施の形態６にかかる制御プログラム開発支援装置によれば、インストラクションリスト等のシーケンス処理用の言語で記述された制御プログラム１を変換して得られた高級プログラミング言語の高級言語変換ファイル４に、元の制御プログラム１の各インストラクションに対応する部分の記述

が挿入されているので、特に、制御プログラムの開発者が、インストラクションリストの記述に不慣れではあるが、Ｃ言語等の高級プログラミング言語について熟知している場合やその逆の場合に有効となり、制御プログラムの開発時間やデバッギング時間の短縮化を図ることができる。

【０１４９】

また、Ｃ言語等の高級プログラミング言語で記述された状態のプログラムファイルに対して修正をおこなった際に、元のインストラクションリスト等の制御プログラムを同時に変更することができるので、これら二者間の整合性を高めることも可能となる。

【０１５０】

実施の形態７.

つぎに、実施の形態７にかかるプログラマブルコントローラについて説明する。実施の形態７にかかるプログラマブルコントローラは、変更前の制御プログラムと変更後の制御プログラムの間の差分データに基づいて生成されたバイナリパッチを受信して記憶し、そのバイナリパッチを実行させることで、実行状態にあるシーケンス処理の所定のタイミングで、実行中の実行コードを新規な実行コードに差し替えるとともに、新規な実行コードによるシーケンス処理を継続することを特徴としている。

【０１５１】

図２２は、実施の形態７にかかるプログラマブルコントローラにおいて、バイナリパッチを利用した実行コード置換処理を説明するための説明図である。まず、プログラマブルコントローラが全く新規な実行コードを受信して実行する処理について説明する。

【０１５２】

図２２（ａ）に示すように、まず、制御プログラム開発支援装置２０によって、実行コード８を生成する。この制御プログラム開発支援装置２０による実行コード生成処理は、従来の制御プログラム開発支援装置または実施の形態１～６にかかる制御プログラム開発支援装置と同様であるので、ここではその説明を省略する。

【０１５３】

そして、プログラマブルコントローラ１０は、上記した実行コード８を図示しない通信インターフェースを介して受け取り、プログラムローダ１２０によって、ワークメモリ等のメモリ１４に読み込む。これにより、プログラマブルコントローラ１０は、実行コード８に従ったシーケンス処理に移行することができる。

【０１５４】

つぎに、実行コード８の元のソースファイルとなる制御プログラム１に変更が生じ、その変更の結果、制御プログラム１’が生成されたとする。特に、この変更は、変更前の制御プログラム１に対して部分的な変更であるとし、全く新たな内容を一から書き下ろした訳ではないとする。この際、まず上述した実行コード生成処理と同様に、制御プログラム１’に対応した実行コード８’が生成される。そして、制御プログラム開発支援装置２０は、保持してあった変更前の実行コード８を読み出し、その実行コード８と変更後の実行コード８’との差分データを差分データ生成ツール１３０によって生成する。

【０１５５】

なお、実行コード８や実行コード８’は、ＡＳＣＩＩコードで表されたテキストデータではなく、バイナリコードで表されたオブジェクトファイルである。よって、これら実行コード間のバイナリデータとしての差異は、元の制御プログラム１および１’間の差異程度の局所的なものになっている。例えば、コンパイラ１００が、変更前の制御プログラム１の各行をオブジェクトファイル中の対応するブロック位置に変換しているだけのものならば、その差異は、何番目のブロック位置に新たにブロックが追加されたのみや、また、何番目のブロック位置が新たにこのブロックに置き換わったのみ程度しかない。

【０１５６】

また、その差異は、各実行コード中のテキストセグメント、データセグメント、ｂｓｓセグメント（初期化されていないセグメント）等の差異でしかない。そのような差異を抽出し、ファイルとして出力するのが、差分データ生成ツール１３０であり、出力された差分データがバイナリパッチ７と呼ばれるものである。

【０１５７】

パッチとは、コンピュータの世界で一般に使われている、ファイルの差分から元データを再現する手法およびその差分データのことであり、一般的にはＡＳＣＩＩコードのいわゆるテキストファイル間の差分に基づいて新規なテキストファイルを生成するものであるが、それらをバイナリデータに対しておこなうようにしたものがバイナリパッチである。

【０１５８】

しかしながら、これらは、コンピュータ上のハードディスクのようないわゆる２次記憶装置上に置かれたファイルに対しておこなうものであり、実行中のメモリ上のプログラムに対しておこなうような手段ではない。ところが、実施の形態７にかかるプログラマブルコントローラでは、そのバイナリパッチ７を、実行中の実行コードに対しておこなう。

【０１５９】

よって、プログラマブルコントローラ１０は、新たに生成された実行コード８’ではなく、それよりもサイズの小さなバイナリパッチ７を、図示しない通信インターフェースを介して受け取り、フラッシュメモリ等の不揮発性メモリに保持する。

【０１６０】

ここで、プログラマブルコントローラ１０上のプログラムローダ１２０は、パッチをあてるパッチャとしても機能し、実行状態にある実行コード８の繰り返しループ中の適当なタイミングで、バイナリパッチデータ７から情報を読み取り、実行中の実行コード８に対してコードの置換処理を実行する。すなわち、上述したように、あるブロック位置へのブロックの挿入や、ブロックの書き換え、または各セグメントに対する挿入・書き換え処理をおこなう。そして、プログラマブルコントローラ１０は、実行コードの書き換えが終了すると、直ちにその新たな実行コードの実行をおこない、シーケンス処理を継続する。

【０１６１】

以上に説明したとおり、実施の形態７にかかるプログラマブルコントローラによれば、制御プログラムのうち変更箇所のみが記述された差分データに基づいて実行中の実行コードを自動的に置換するとともに、置換された新たな実行コード

によるシーケンス処理を継続するので、従来のように、変更後の実行コードをまるごとプログラマブルコントローラ１０に保持せずとも、その実行コードよりもサイズの小さいバイナリパッチ７を保持すればよく、メモリ資源の縮小化を図ることができる。

【０１６２】

なお、上述した差分データによって制御プログラムの変更に対応する概念は、インタプリタ型のプログラマブルコントローラにおいて、ラダーダイヤグラムやインストラクションリストに対しても適用することができる。

【０１６３】

実施の形態８．

つぎに、実施の形態８にかかる制御プログラム開発支援装置について説明する。実施の形態８にかかる制御プログラム開発支援装置は、コンパイルすることにより得られる実行コードに、コンパイル前の制御プログラムのファイルを圧縮した形で挿入し、実行コードと制御プログラムの一元化を図ることを特徴としている。

【０１６４】

図２３は、実施の形態８にかかる制御プログラム開発支援装置における実行コード生成処理を説明するための説明図である。まず、制御プログラム開発支援装置は、インストラクションリスト等の制御プログラム１に対し、コンバータ１０１ａにより、Ｃ言語等の高級プログラミング言語に変換し、その変換により得られたＣ言語表現のプログラムファイル４ａをＣコンパイラ１０２に投入する。そして、Ｃコンパイラ１０２は、Ｃ言語表現のプログラムファイル４ａをコンパイルしてオブジェクトファイル８ａを出力する。

【０１６５】

一方で、制御プログラム開発支援装置は、ファイル圧縮ツール１４０によって、制御プログラム１を圧縮して圧縮ファイル９９を生成する。なお、上記したファイル圧縮ツール１４０としては、例えばｇｚｉｐ、ｂｚｉｐ２、ｌｈａ等を利用することができる。このようなツールにより圧縮された圧縮ファイル９９は、ＡＳＣＩＩコードのテキスト形式ではなく、バイナリデータとなっている。

【０１６６】

つづいて、コンバータ１０１ｂによって、圧縮ファイル９９をＣ言語表現のプログラムファイル４ｂに変換する。このように、圧縮ファイル９９をコンバータ１０１ｂによってＣ言語表現に変換する理由は、圧縮された制御プログラムのソースコードを単なる添付ファイルとしてでなく、最終的に生成される実行コードの一部に含めるためである。

【０１６７】

図２４は、制御プログラム１の内容であるソースファイルの例を示す図である。また、図２５は、圧縮ファイル９９を変換して得られたＣ言語表現の例を示す図である。図２４に示すようなインストラクションリストで表された制御プログラム１は、上述したように、ファイル圧縮ツール１４０によって、バイナリデータとして表された圧縮ファイル９９に変換される。

【０１６８】

その圧縮ファイル９９が、例えば先頭のデータから“０１ｈ，０２ｈ，０３ｈ．．．”と並んでいたならば、コンバータ１０１ｂによって、１バイトデータ単位の配列｛０ｘ０１，０ｘ０２，０ｘ０３，．．．．｝のように変換される。なお、図２５に示す配列データは、図２４に示したソースファイルに対する変換結果である。

【０１６９】

このように、圧縮ファイルをＣコンパイラ１０２で解釈できる形に変換してＣ言語表現のプログラムファイル４ｂを生成している。そして、Ｃコンパイラ１０２は、Ｃ言語表現のプログラムファイル４ａと同様に、Ｃ言語表現のプログラムファイル４ｂをコンパイルしてオブジェクトファイル８ｂを出力する。

【０１７０】

Ｃコンパイラ１０２により生成された二つのオブジェクトファイル８ａおよび８ｂは、リンカ１５０によってリンクされ、最終的に、プログラマブルコントローラにおいて直接実行可能な一つの実行コード９０を得ることができる。すなわち、実行コード９０には、実際にシーケンス処理に寄与する部分と制御プログラム１のソースコードの情報とによって構成されている。

【０１７１】

このようにして得られた実行コード９０は、一般に、プログラマブルコントローラの不揮発性メモリ等に保持されるが、例えば図２５に示しているように、制御プログラム１を圧縮したデータは、＿ＩｎｐｕｔＳｏｕｒｃｅという名前の配列であり、そのファイル名がｓａｍｐｌｅ．ｉｌ，ｇｚであるとして実行コード内に含有されているため、その名前を元に、圧縮ファイル９９を抽出することが可能である。そして、圧縮ファイル９９を抽出した後は、ファイル圧縮ツール１４０において一般に備わっている展開機能やｇｕｎｚｉｐ、ｂｕｎｚｉｐ２等のような圧縮形式に対応する展開ツールを用いることにより、元の制御プログラム１を復元することが可能になる。

【０１７２】

以上に説明したとおり、実施の形態８にかかる制御プログラム開発支援装置によれば、実際のシーケンス処理に寄与する部分とインストラクションリスト等の制御プログラム１のソースコード情報とが一体となった実行コードを生成するので、制御プログラムと実行コードとを一元化してプログラマブルコントローラに保持することができ、開発やデバッグ時に必要となる制御プログラムを消失した場合や現場のプログラマブルコントローラから直接制御プログラムを操作したい場合において、実行コードからその制御プログラムを取り出すことができるとともに、実行コードと制御プログラムの整合性を図ることもできる。

【０１７３】

実施の形態９．

つぎに、実施の形態９にかかるプログラマブルコントローラおよび制御プログラム開発支援装置について説明する。実施の形態１にかかるプログラマブルコントローラは、制御プログラムを実行コードにインタプリーティングする際に、制御プログラムのうち出現頻度の高い命令順にパターンマッチングをおこなうことを特徴としており、実施の形態９にかかる制御プログラム開発支援装置は、制御プログラムを実行コードにコンパイルする際に、制御プログラムのうち出現頻度の高い命令順にパターンマッチングをおこなうことを特徴としている。

【０１７４】

図２６は、実施の形態９にかかる制御プログラム開発支援装置におけるコンパイル処理を説明するための説明図である。図２６は、特に、ユーザが記述したインストラクションリスト等の制御プログラムである入力ソースファイル１を変換し、プログラマブルコントローラにおいて直接実行可能な実行コードとなる出力ファイル７０を生成するシステムを示している。

【０１７５】

一般に、インストラクションリストは、ＬＤ命令やＯＵＴ命令が基本的に多く使用されることや、ユーザが同じ内容の処理をおこなう場合に、ある特定の命令を多用する等の理由により、命令の出現回数に多くの偏りを有している。また、多くのプログラムが、過去のプログラムの実績に基づく信頼性から多く流用されるため、それにより使用する命令の偏りも存在する。

【０１７６】

そこで、コンパイラに相当するプログラム変換ツール１６０では、パターン検出・並べ替え部１７０を備えている。パターン検出・並べ替え部１７０には、パターン出現カウント部１７１とパターン比較並べ替え部１７２を備えている。パターン出現カウント部１７１は、各プログラム変換ツール１６０で変換する際に、その命令の出現回数をカウントし、それをパターン出現頻度保存ファイル６０に保存する。

【０１７７】

パターン比較並べ替え部１７２は、パターン出現頻度保存ファイル６０から出現回数の多い命令の順番に、パターンマッチ部１８０の第１～４のパターン比較１８１ａ～１８１ｄをパターンマッチングの順番の先の方に並び替える。もちろん、各パターン比較に対する第１～４のアクション実行部１８２ａ～１８２ｄは連動する。これにより、出現頻度の多い命令ほど早くパターンマッチするため、処理を完了するまでの時間は短くなる。パターン比較並べ替え部１７２が並べ替えるタイミングは、命令一つ一つに対し、パターンマッチ部１８０から抜けて、行検出部１６１に戻るときでも良いし、プログラム変換ツール１６０が起動される際のタイミングでもよい。

【０１７８】

つぎに、この制御プログラム開発支援装置のコンパイル動作について説明する。図２７は、実施の形態９にかかる制御プログラム開発支援装置におけるコンパイル動作を示すフローチャートである。

【０１７９】

まず、制御プログラム開発支援装置は、プログラム変換ツール１６０に対して、入力ソースファイル５０を入力する。ここで、入力ソースファイル５０は、インストラクションリスト

ＬＤ　Ｘ０

ＡＮＤ　Ｘ１０

ＯＵＴ　Ｙ０

等が記述されているものとする。

【０１８０】

まず、制御プログラム開発支援装置は、後述するパターン出現頻度保存ファイル６０を読み込み（ステップＳ４０１）、パターン比較並び替え部１７２によって、出現頻度の多い命令順を認識する（ステップＳ４０２）。これにより、パターン比較並び替え部１７２は、パターンマッチ部１８０内の第１～４のパターン比較部１８１ａ～１８１ｄおよびそれらに対応する第１～４のアクション実行部１８２ａ～１８２ｄの並び替えをおこなう（ステップＳ４０３）。

【０１８１】

そして、プログラム変換ツール１６０中において、行検出部１６１が入力ソースファイル５０のうちの１行を検出し（ステップＳ４０４）、検出した１行が入力ソースファイル５０の終了位置でないならば（ステップＳ４０５否定）、それを意味解釈部１６２に渡す。

【０１８２】

上記のインストラクションリスト例では、まず、“ＬＤ　Ｘ０”が最初に意味解釈部１６２に渡される。意味解釈部１６２では、これがＬＤ命令であることを解釈し、ＬＤ命令の実行部を検索するためにパターンマッチ部１８０にＬＤ命令の実行部の検索をさせる（ステップＳ４０８）。

【０１８３】

パターンマッチ部１８０では、内部的に実装されている各種パターン、上記例では、命令“ＬＤ”のアクションがどれかを、順番に第１のパターン比較１８１ａ、第２のパターン比較１８１ｂ、第３のパターン比較１８１ｃ、第４のパターン比較１８１ｄ、というようにパターンマッチをおこない（ステップＳ４０９）、例えば、第４のパターン比較１８１ｄがＬＤ命令に関する内容ならば、その動作である第４のアクション実行部１８２ｄを実行する（ステップＳ４１０）。第４のアクション実行部１８２ｄは、ＬＤ命令に対応した実行コード記述を出力ファイル７０に出力し、パターン出現カウント部１７１により、その命令の出現頻度がカウントされる（ステップＳ４１１）。

【０１８４】

この処理が終わると、再度、ステップＳ４０４に戻り、上記例では、つぎの行である“ＡＮＤ　Ｘ１０”に対して同じ処理を繰り返す。一方、検出した１行が入力ソースファイル５０の終了位置であるならば（ステップＳ４０５肯定）、出力ファイル７０への書き込みを終了し（ステップＳ４０６）、パターン出現頻度保存ファイル６０に、パターン出現カウント部１７１によってカウントされた結果を更新データとして保存する（ステップＳ４０７）。

【０１８５】

以上に説明したとおり、実施の形態９にかかる制御プログラム開発支援装置によれば、制御プログラムを実行コードにコンパイルする際に、制御プログラムのうち出現頻度の高い命令順にパターンマッチングをおこなうので、コンパイルの際に、制御プログラムの各命令に適合する実行コードを迅速に発見でき、制御プログラムをコンパイルするための待ち時間を減少させることが可能になる

【０１８６】

なお、上述したプログラム変換ツール１６０の動作は、インストラクションリスト等の制御プログラムをＣ言語等の高級プログラミング言語で記述されたプログラムファイルに変換するコンバータに適用することも可能である。すなわち、上述した説明において、出力ファイル７０に書き込まれる内容は、実行コードではなく、高級プログラミング言語の命令表現となる。

【０１８７】

さらに、上述したプログラム変換ツール１６０の動作は、インタプリタ型のプログラマブルコントローラにも適用することができる。この場合、入力ソースファイル５０は、インストラクションリストでも、それを変換した中間コードでもよく、第１～４のアクション実行部１８２ａ～１８２ｄで実行されるアクションは、実際のプログラマブルロジック処理になる。

【０１８８】

なお、以上に説明した実施の形態１～６、８および９にかかる制御プログラム開発支援装置によって実現される機能と、実施の形態１、７および９にかかるプログラマブルコントローラによって実現される機能は、制御プログラムの開発とともにシーケンス処理も可能とする、いわゆるソフトウェアプログラマブルコントローラに搭載することも可能である。

【０１８９】

【発明の効果】

以上、説明したとおり、この発明によれば、高速化実装手段を搭載した汎用マイクロプロセッサを備えるとともに、受信した実行コードが、その汎用マイクロプロセッサの高速化実装手段を用いて実行されるように最適化されているので、マイクロプロセッサがＡＳＩＣとして提供される従来のプログラマブルコントローラと比較して、より高速にシーケンス処理を実行することができるという効果を奏する。

【０１９０】

つぎの発明によれば、シーケンス制御用の言語で記述された制御プログラムを、高速化実装手段を搭載した汎用マイクロプロセッサで実行できる実行コードにコンパイルするので、パーソナルコンピュータ等に使用される汎用マイクロプロセッサを搭載したプログラマブルコントローラにおいて、シーケンス処理を実行させることができるとともに、コンパイル手段を変更することにより、様々な汎用マイクロプロセッサに対応した実行コードを得ることができるので、制御プログラムを変更せずとも様々なプログラマブルコントローラに対応することができ、過去の制御プログラムの資産の有効活用、制御プログラムの管理の一元化および制御プログラムの開発時間の効率化を図ることが可能になるという効果を奏す

る。

【０１９１】

つぎの発明によれば、制御プログラムを複数のブロックに分割して、分割したブロック毎にコンパイルして最終的な実行コードを得ることができるので、長大な一本の制御プログラムについて変更が生じた場合であっても、従来のように制御プログラム全体を再コンパイルすることなく、変更の生じた一部分だけを再コンパイルすることで実行コードを生成することができ、制御プログラムのデバッグや保守作業に要する時間を短縮することができるという効果を奏する。

【０１９２】

つぎの発明によれば、制御プログラムを複数のブロックに分割して、分割したブロックのそれぞれを汎用コンピュータ用高級言語に変換し、各ブロックに対応する汎用コンピュータ用高級言語のプログラムをコンパイルして最終的な実行コードを得ることができるので、長大な一本の制御プログラムについて変更が生じた場合であっても、従来のように制御プログラム全体を再コンパイルすることなく、変更の生じた一部分だけを再コンパイルすることで実行コードを生成することができるとともに、コンパイラとして従来の汎用コンピュータ用高級言語用のコンパイラを使用することができ、制御プログラムのデバッグや保守作業に要する時間を短縮することができるという効果を奏する。

【０１９３】

つぎの発明によれば、パーソナルコンピュータ等に使用される汎用マイクロプロセッサを搭載したプログラマブルコントローラに対して実行コードを生成するので、マイクロプロセッサがＡＳＩＣとして提供される従来のプログラマブルコントローラと比較して、パイプライン、スーパスカラ、アウトオブオーダ等の高速処理によって、高速にシーケンス処理を実行することができるという効果を奏する。

【０１９４】

つぎの発明によれば、制御プログラム分割手段が、ラダーダイヤグラム内の所定のラングの切れ目において複数のブロックに分割し、当該ブロック毎にプログラムファイルを生成するので、入力から出力までの一連の回路ブロックを単位と

してプログラムファイルを管理することができるとともに、コンパイル時における入出力デバイスのレジスタの最適化が分割によって阻害されるという事態を低減させることができ、特に、高速化実装手段を搭載したマイクロプロセッサに対しては、複数のラングの集まりで一つのブロックを構成することで、アウトオブオーダのコードを出力するコンパイラの最適化機能を活用することができるという効果を奏する。

【０１９５】

つぎの発明によれば、制御プログラム分割手段が、ラダーダイヤグラム内の所定のラングの切れ目であってかつジャンプ命令に対するジャンプ先となる位置において複数のブロックに分割し、当該ブロック毎にプログラムファイルを生成するので、コンパイラの最適化に影響を与えないとともに、分割後のプログラムファイルをサブルーチンとして管理することができ、制御プログラムのデバッグや保守作業を容易にするという効果を奏する。

【０１９６】

つぎの発明によれば、制御プログラム分割手段が、ラダーダイヤグラムにおいて共通の入力デバイスまたは出力デバイスに対する命令を含んだラングのすべてまたは一部を抽出し、抽出したラングのすべてまたは一部を一つのブロックとして複数のブロックに分割し、当該ブロック毎にプログラムファイルを生成するので、特定のデバイスに対する制御プログラムの変更が生じた際に、その部分に該当するプログラムファイルのみを変更して再コンパイルして実行コードを得ることができ、制御プログラムのデバッグや保守作業に要する時間を短縮することができるという効果を奏する。

【０１９７】

つぎの発明によれば、制御プログラムを、コンパイルする前段において、一旦、参照されない変数の排除、冗長コードの排除、並べ替え等をおこなうことによって最適なコード体系に再構築するので、最適化されたコードにより構成された制御プログラムをコンパイル手段に提供することができ、コンパイルに要する時間の低減と実行時の速度の向上を図ることができるという効果を奏する。

【０１９８】

つぎの発明によれば、処理時間が既知であるサンプルプログラムと実行コードに対応する制御プログラムとを対応付けた対応表から、シーケンス処理実行時間を見積もることができるので、従来は、コンパイラの最適化によって生成された実行コードの実際の処理時間とそれに対応する制御プログラム内の各インストラクションとを正確に対応付けて知得することが困難であったが、それが可能となるという効果を奏する。

【０１９９】

つぎの発明によれば、インストラクションリストとそのインストラクションリストを変換することで得られた高級プログラミング言語の制御プログラムとの実行部分を対応付けてステップ実行できるとともにその部分を表示することができるので、インストラクションリストをコンパイルして得られた実行コードに基づいて動作するプログラマブルコントローラにおいても、ユーザが記述したインストラクションリストのステップ実行レベルで動作確認がおこなえるようになり、制御プログラムの開発およびデバッグ作業を容易にすることができるという効果を奏する。

【０２００】

つぎの発明によれば、現在実行されている実行コードと新たな実行コードとの差分データに基づいて、現在実行中の実行コードを、所定のタイミングで新たな実行コードに変更し、変更後の実行コードを継続的に実行させるので、実行コードに変更が生じた場合であっても、プログラマブルコントローラには新たな実行コードではなく、それよりもサイズの小さな差分データをメモリにダウンロードすることで実行コードの変更と実行状態の継続をさせることができ、プログラマブルコントローラのメモリ資源を多く要しないという効果を奏する。

【０２０１】

つぎの発明によれば、制御プログラムと実行コードとを一元化してプログラマブルコントローラに保持するので、開発やデバッグ時に必要となる制御プログラムを消失した場合や現場のプログラマブルコントローラから直接制御プログラムを操作したい場合において、実行コードからその制御プログラムを取り出すことができるとともに、実行コードと制御プログラムの整合性を図ることもできると

いう効果を奏する。

【０２０２】

つぎの発明によれば、制御プログラムと実際に実行される実行コードとを一元化してあらためて実行コードとして生成するので、開発やデバッグ時に必要となる制御プログラムを消失した場合や現場のプログラマブルコントローラから直接制御プログラムを操作したい場合において、実行コードからその制御プログラムを取り出すことができるとともに、実行コードと制御プログラムの整合性を図ることもできるという効果を奏する。

【０２０３】

つぎの発明によれば、制御プログラムを実行コードにインタプリーティングする際に、制御プログラムのうち出現頻度の高い命令順にパターンマッチングをおこなうので、インタプリーティングの際に、制御プログラムの各命令に適合する実行コードを迅速に発見でき、制御プログラムの実行時間を減少させることが可能になるという効果を奏する。

【０２０４】

つぎの発明によれば、制御プログラムを実行コードにコンパイルする際に、制御プログラムのうち出現頻度の高い命令順にパターンマッチングをおこなうので、コンパイルの際に、制御プログラムの各命令に適合する実行コードを迅速に発見でき、制御プログラムをコンパイルするための待ち時間を減少させることが可能になるという効果を奏する。

【０２０５】

つぎの発明によれば、制御プログラムの開発とともにシーケンス処理も可能とする、いわゆるソフトウェアプログラマブルコントローラにおいて、上記した制御プログラム開発支援装置と上記したプログラマブルコントローラの各機能を実現することができるという効果を奏する。

【図面の簡単な説明】

【図１】　実施の形態１にかかるプログラマブルコントローラの概略構成を示すブロック図である。

【図２】　実施の形態１にかかる制御プログラム開発支援装置において、実

行コードの生成を説明するための説明図である。

【図３】　実施の形態２にかかる制御プログラム開発支援装置において、制御プログラムの分割をともなう実行コード生成処理を説明するための説明図である。

【図４】　ラダーダイヤグラムの例を示す図である。

【図５】　実施の形態２にかかる制御プログラム開発支援装置において、ステップ数と分割後ファイルの対応関係を示した表である。

【図６】　実施の形態２にかかる制御プログラム開発支援装置において、ラング分割方法の処理を示したフローチャートである。

【図７】　実施の形態２にかかる制御プログラム開発支援装置において、ジャンプ先ラベル名と分割後ファイルの対応関係を示した表である。

【図８】　実施の形態２にかかる制御プログラム開発支援装置において、ジャンプ先分割方法の処理を示したフローチャートである。

【図９】　実施の形態２にかかる制御プログラム開発支援装置において、機器分割方法の処理を示したフローチャートである。

【図１０】　実施の形態２にかかる制御プログラム開発支援装置において、制御プログラムに修正が生じた際の実行コード生成処理を説明するための説明図である。

【図１１】　実施の形態３にかかる制御プログラム開発支援装置において、制御プログラムの分割と高級プログラミング言語への変換とをともなう実行コード生成処理を説明するための説明図である。

【図１２】　実施の形態３にかかる制御プログラム開発支援装置において、高級プログラミング言語に一旦変換する場合の制御プログラムに修正が生じた際の実行コード生成処理を説明するための説明図である。

【図１３】　実施の形態３にかかる制御プログラム開発支援装置において、コンバータが制御プログラムの分割と高級プログラミング言語への変換をおこなう場合の実行コード生成処理を説明するための説明図である。

【図１４】　実施の形態３にかかる制御プログラム開発支援装置において、コンバータが制御プログラムの分割と高級プログラミング言語への変換をおこな

う場合に制御プログラムに修正が生じた際の実行コード生成処理を説明するための説明図である。

【図１５】　実施の形態４にかかる制御プログラム開発支援装置の実行コード生成処理を説明するための説明図である。

【図１６】　実施の形態５にかかる制御プログラム開発支援装置において、サンプルプログラムとそのサイズとその処理時間とを表した表である。

【図１７】　実施の形態６にかかる制御プログラム開発支援装置において、制御プログラムの開発環境および実行環境を説明するための説明図である。

【図１８】　実施の形態６にかかる制御プログラム開発支援装置におけるデバッギング作業を説明するための説明図である。

【図１９】　実施の形態６にかかる制御プログラム開発支援装置において、サンプリングトレース機能を説明するための説明図である。

【図２０】　実施の形態６にかかる制御プログラム開発支援装置において、ステータスラッチ機能を説明するための説明図である。

【図２１】　実施の形態６にかかる制御プログラム開発支援装置におけるデバッギング手法を説明するための説明図である。

【図２２】　実施の形態７にかかるプログラマブルコントローラにおいて、バイナリパッチを利用した実行コード置換処理を説明するための説明図である。

【図２３】　実施の形態８にかかる制御プログラム開発支援装置における実行コード生成処理を説明するための説明図である。

【図２４】　実施の形態８にかかる制御プログラム開発支援装置において、制御プログラムの内容であるソースファイルの例を示す図である。

【図２５】　実施の形態８にかかる制御プログラム開発支援装置において、圧縮ファイルを変換して得られたＣ言語表現の例を示す図である。

【図２６】　実施の形態９にかかる制御プログラム開発支援装置におけるコンパイル処理を説明するための説明図である。

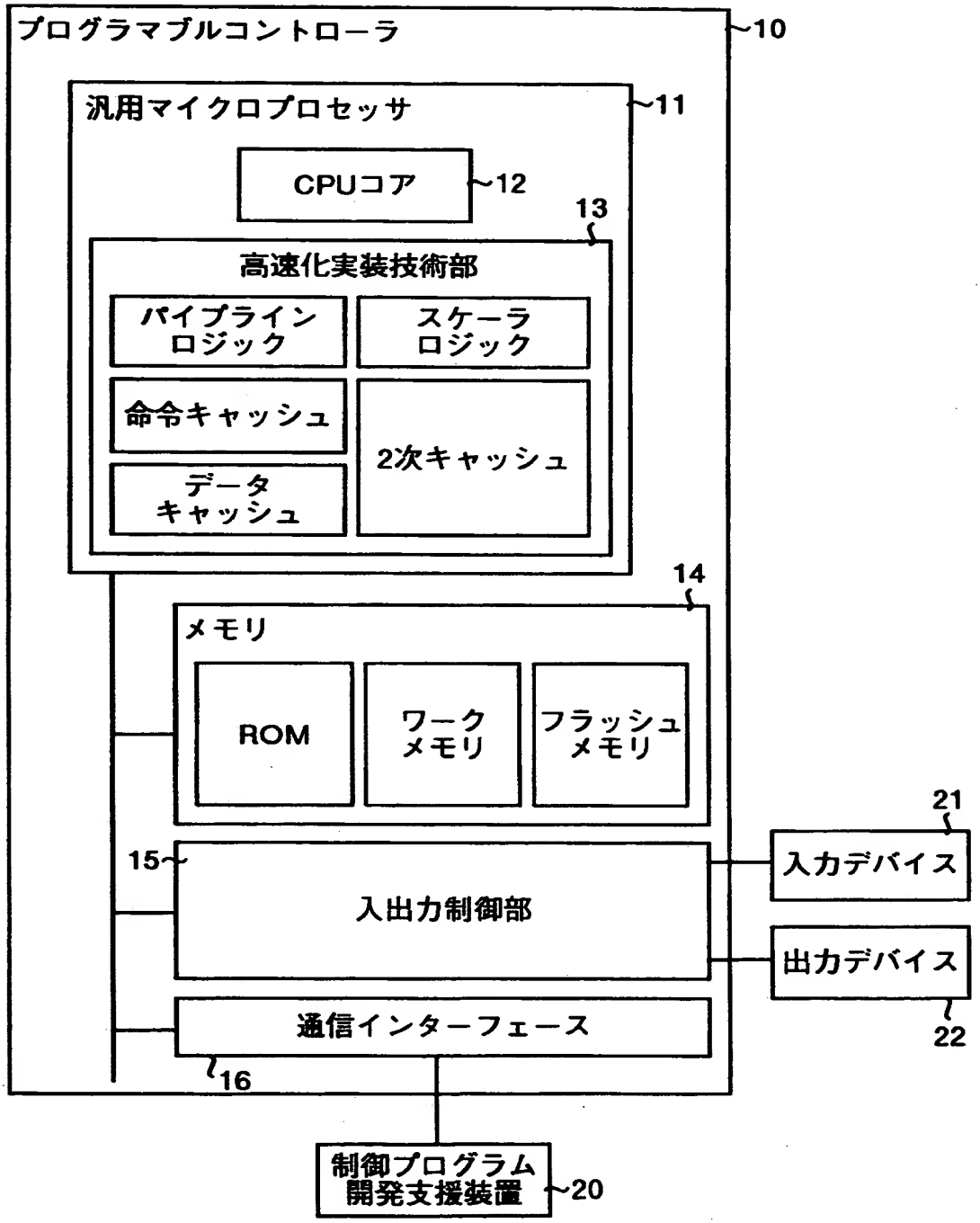
【図２７】　実施の形態９にかかる制御プログラム開発支援装置におけるコンパイル動作を示すフローチャートである。

【符号の説明】

１　制御プログラム、３－１～３－Ｎ　分割プログラム、４－１～４－Ｎ　Ｃ言語表現、８，８’，９　実行コード、１０　プログラマブルコントローラ、１１　汎用マイクロプロセッサ、１２　ＣＰＵコア、１３　高速化実装技術部、１４　メモリ、１５　入出力制御部、１６　通信インターフェース、２０　制御プログラム開発支援装置、２１　入力デバイス、２２　出力デバイス、３０　開発環境、４０　実行環境、１００，１０８　コンパイラ、１０１，１０１ａ，１０１ｂ，１０３　コンバータ、１０２　Ｃコンパイラ、１０６　エディタ、１０７，１５０　リンカ、１１０　デバッガ、１４０　ファイル圧縮ツール、１６０　プログラム変換ツール、１６１　行検出部、１６２　意味解釈部、１７０　パターン検出・並べ替え部。

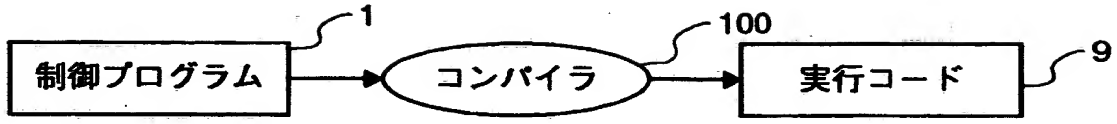
【書類名】　　図面

【図１】

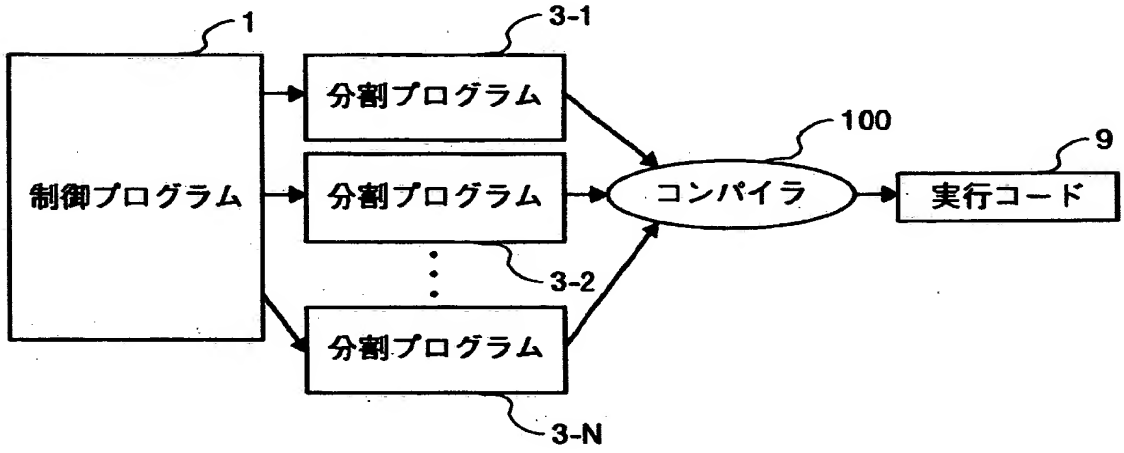


出証特２０００－３０８４７４６

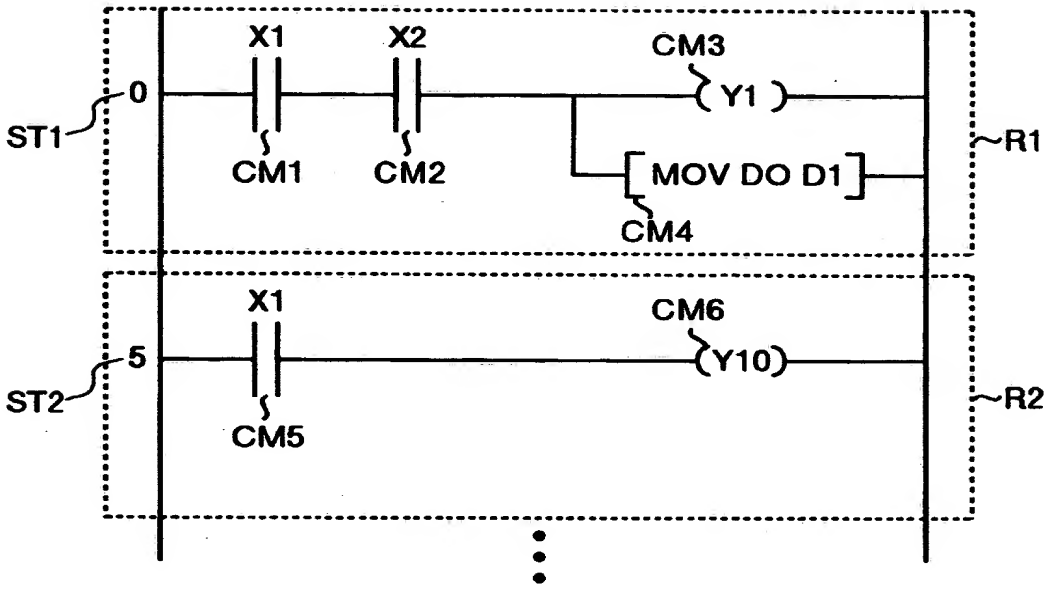
【図２】


1
制御プログラム
コンパイラ
100
実行コード
9

【図３】


1
制御プログラム
3-1
分割プログラム
分割プログラム
3-2
分割プログラム
3-N
100
コンパイラ
9
実行コード

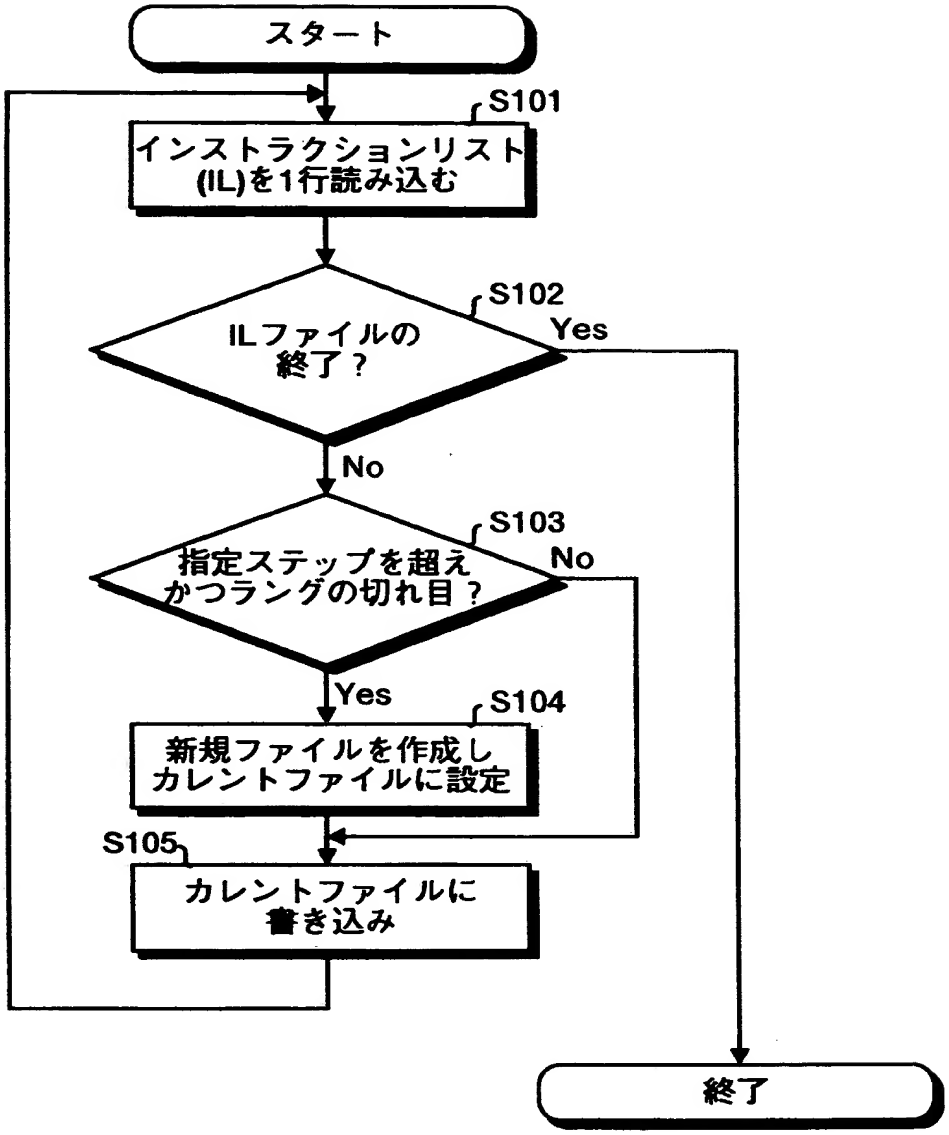
【図４】



【図５】

ステップ数	分割後ファイル名
0～99	Control1.il
100～199	Control2.il
・ ・ ・	・ ・ ・

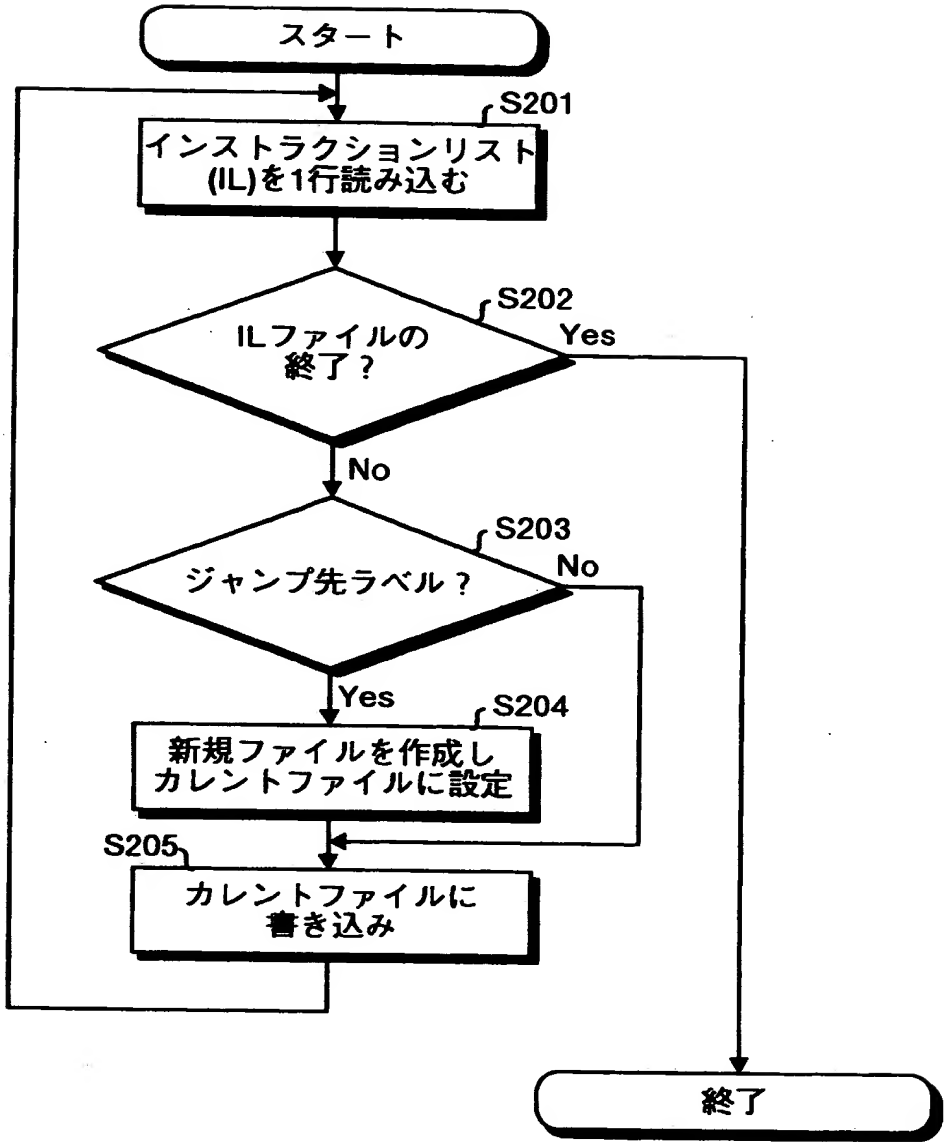
【図６】


スタート
S101
インストラクションリスト(IL)を1行読み込む
S102
ILファイルの終了？
Yes
No
S103
指定ステップを超えかつラングの切れ目？
No
Yes
S104
新規ファイルを作成しカレントファイルに設定
S105
カレントファイルに書き込み
終了

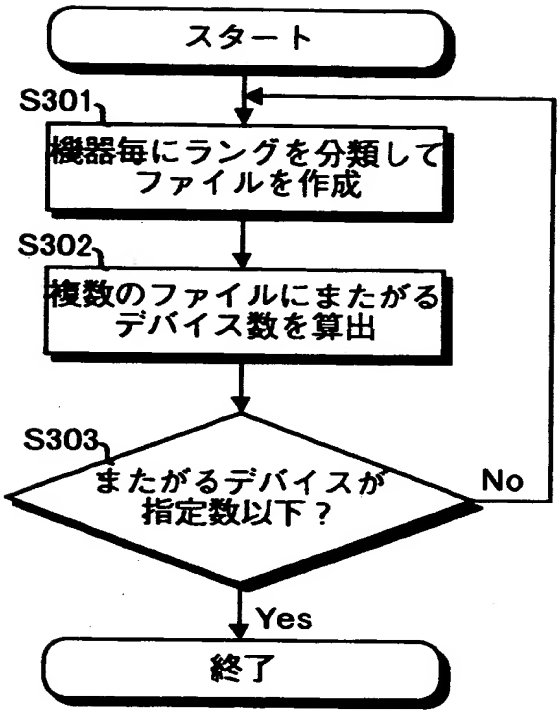
【図７】

ジャンプ先ラベル名	分割後ファイル名
P 1	Control1.il
P 2	Control2.il
•	•
•	•
•	•

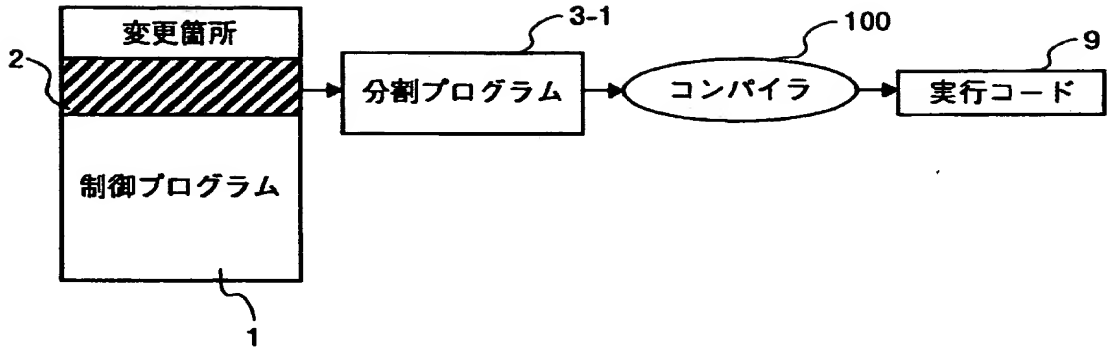
【図８】


スタート
S201
インストラクションリスト(IL)を1行読み込む
S202
ILファイルの終了？
Yes
No
S203
ジャンプ先ラベル？
No
Yes
S204
新規ファイルを作成しカレントファイルに設定
S205
カレントファイルに書き込み
終了

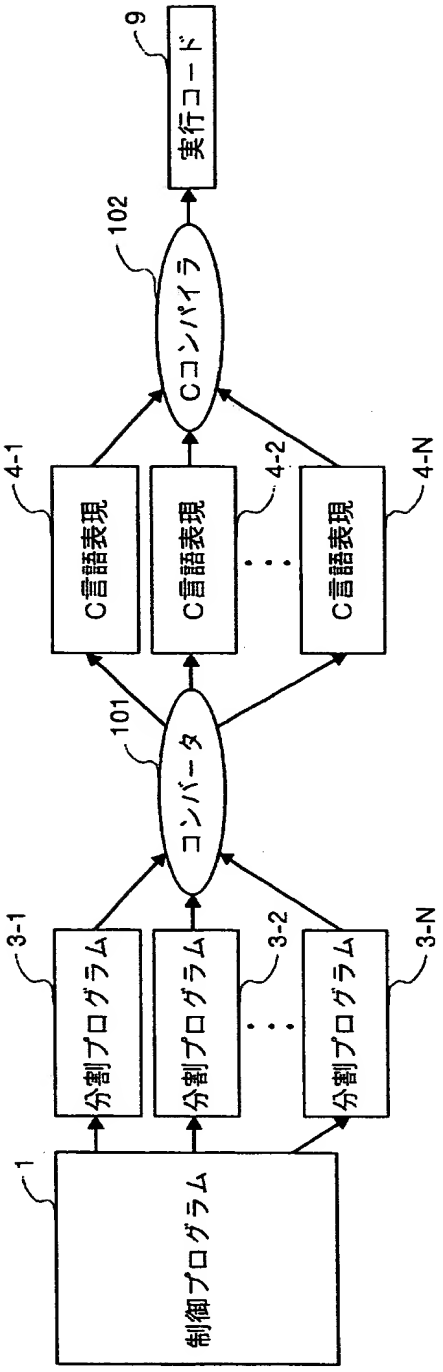
【図９】


スタート
S301
機器毎にラングを分類して
ファイルを作成
S302
複数のファイルにまたがる
デバイス数を算出
S303
またがるデバイスが
指定数以下？
No
Yes
終了

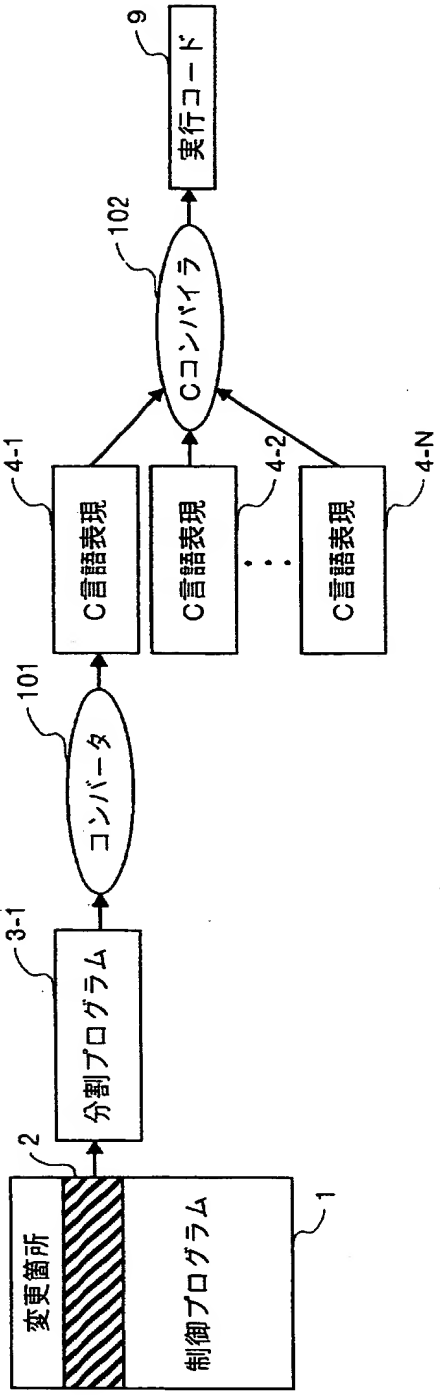
【図１０】



【図１１】

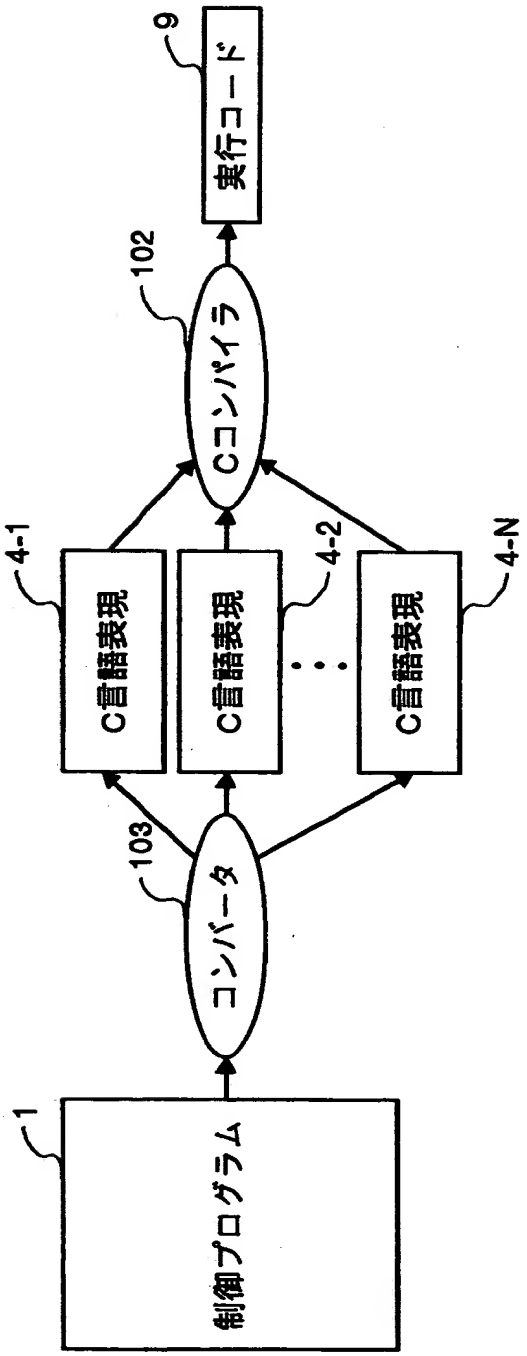

9
実行コード
102
Cコンパイラ
4-1
C言語表現
4-2
C言語表現
4-N
C言語表現
101
コンパータ
3-1
分割プログラム
3-2
分割プログラム
3-N
分割プログラム
1
制御プログラム

【図１２】

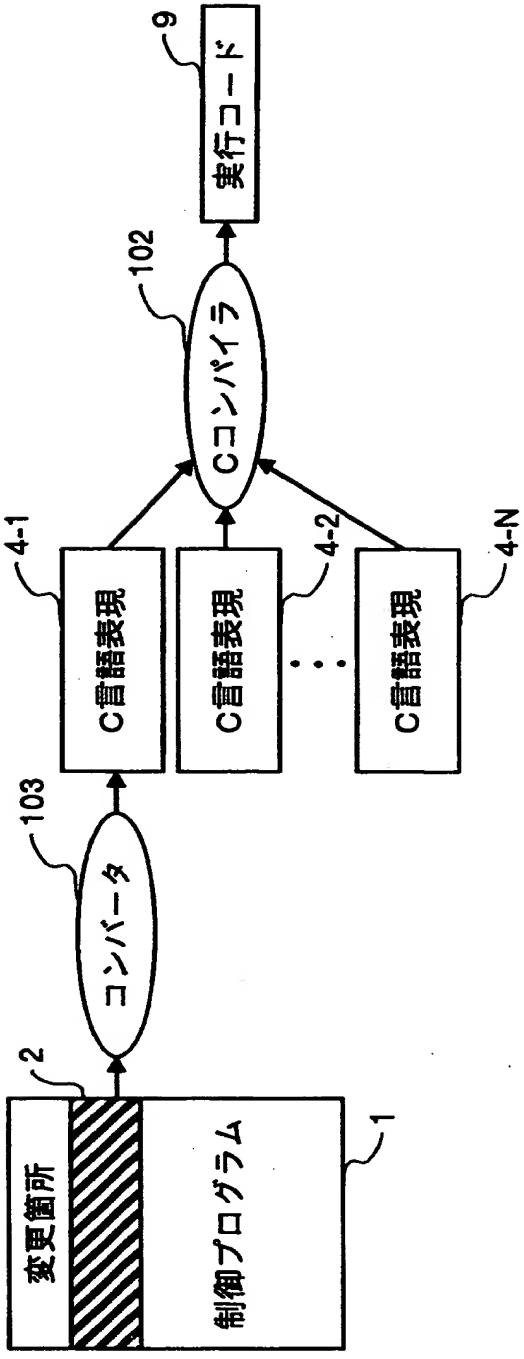


出証特２０００－３０８４７４６

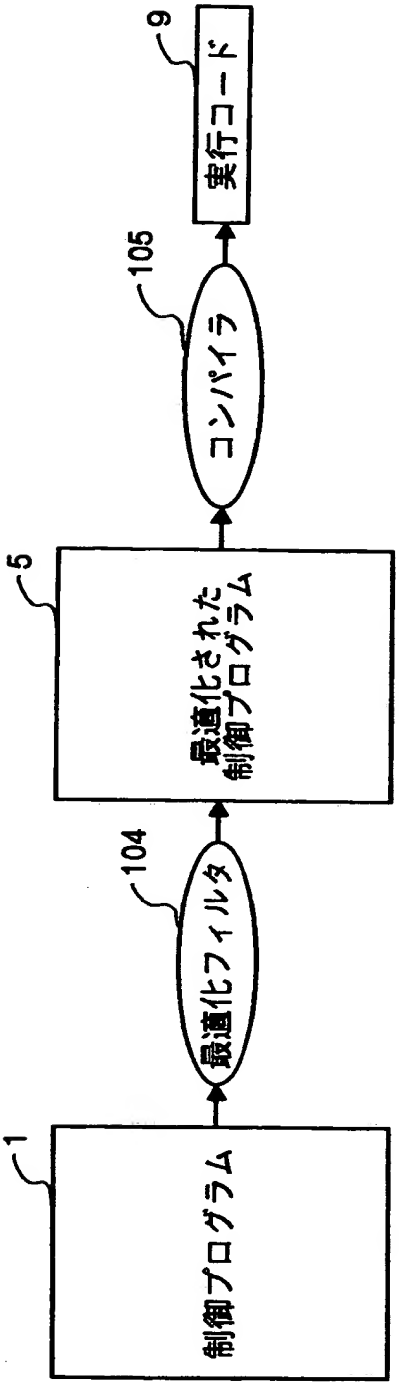
【図１３】


9
実行コード
102
Cコンパイラ
4-1
C言語表現
4-2
C言語表現
4-N
C言語表現
103
コンバータ
1
制御プログラム

【図１４】


9
実行コード
102
Cコンパイラ
4-1
C言語表現
4-2
C言語表現
4-N
C言語表現
103
コンパータ
2
変更箇所
制御プログラム
1

【図１５】

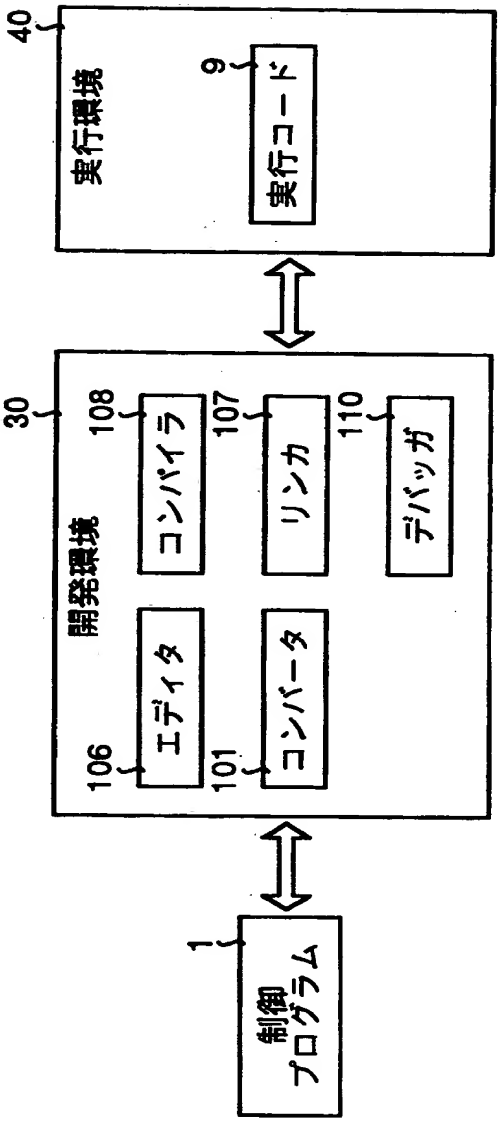


【図１６】

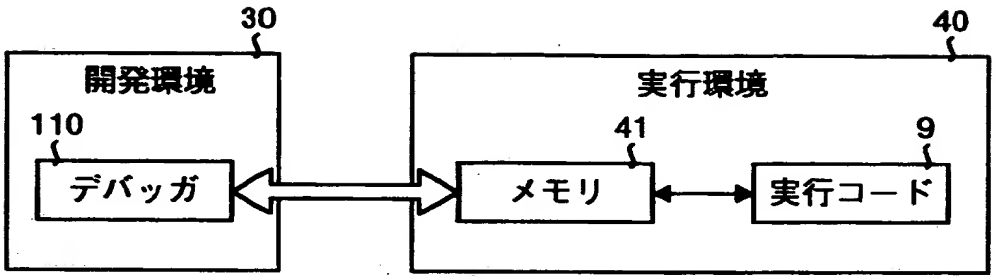
インストラクション	サイズ（ステップ数）	処理時間（msec）
LD,OUT	1K	○○
LD,AND,OUT	1K	△△
⋮	⋮	⋮
LDI,AND,OUT	10K	□□
⋮	⋮	⋮

出証特２０００－３０８４７４６

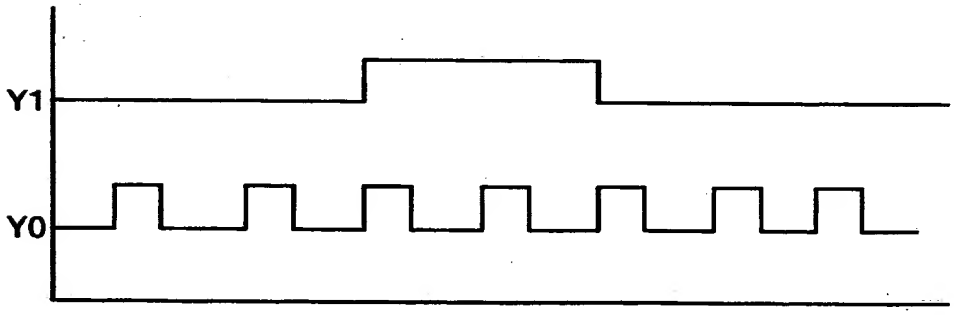
【図１７】


40
実行環境
9
実行コード
30
開発環境
108
コンパイラ
107
リンカ
110
デバッガ
106
エディタ
101
コンバータ
1
制御
プログラム

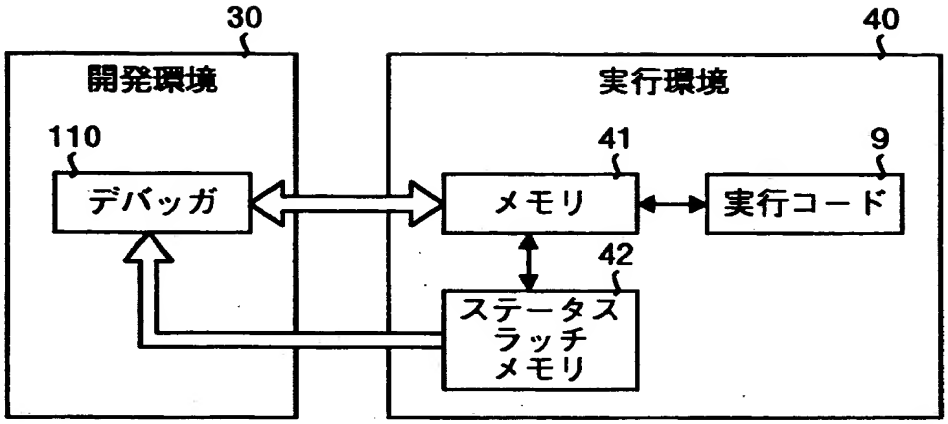
【図１８】



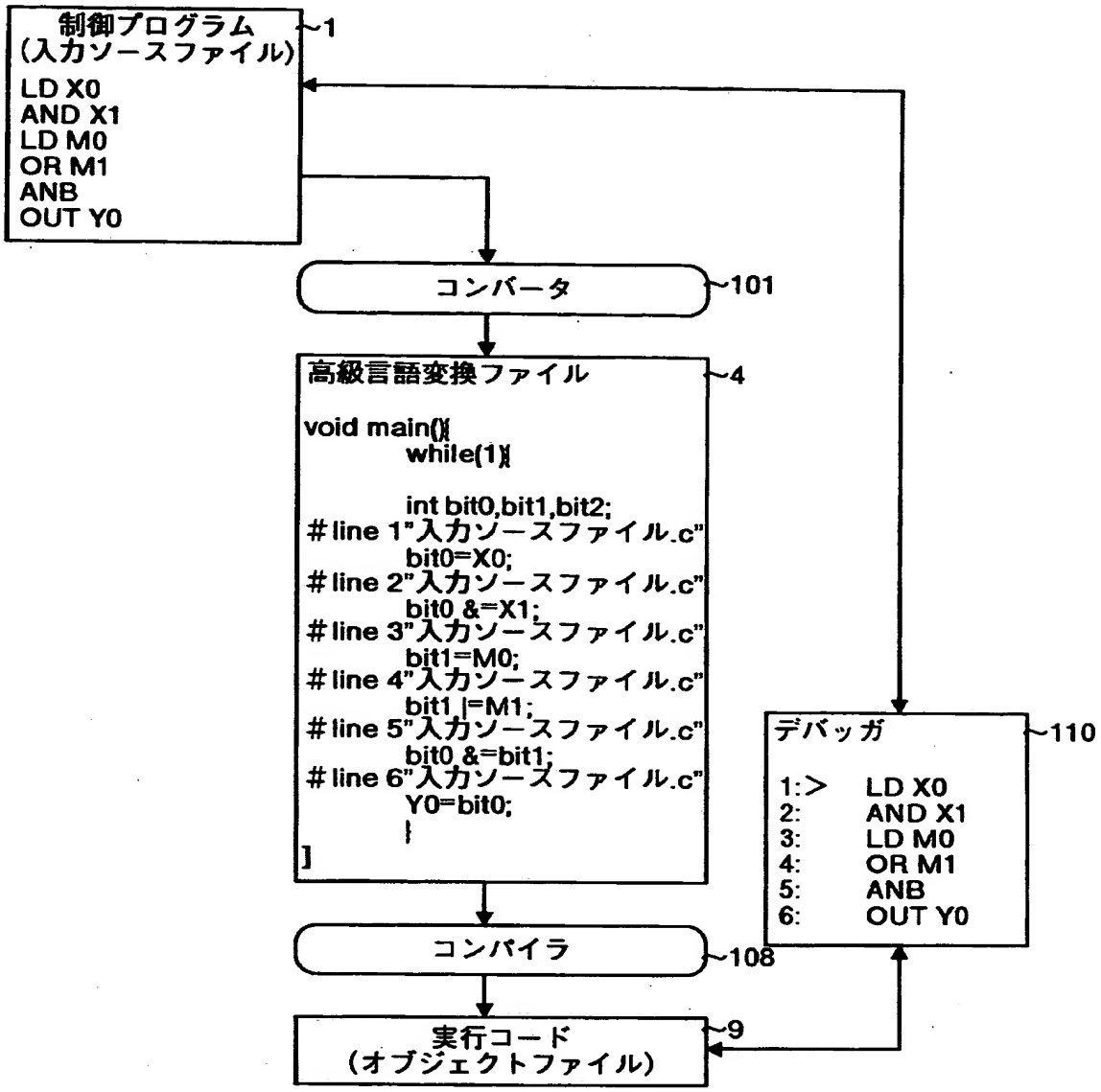
【図１９】



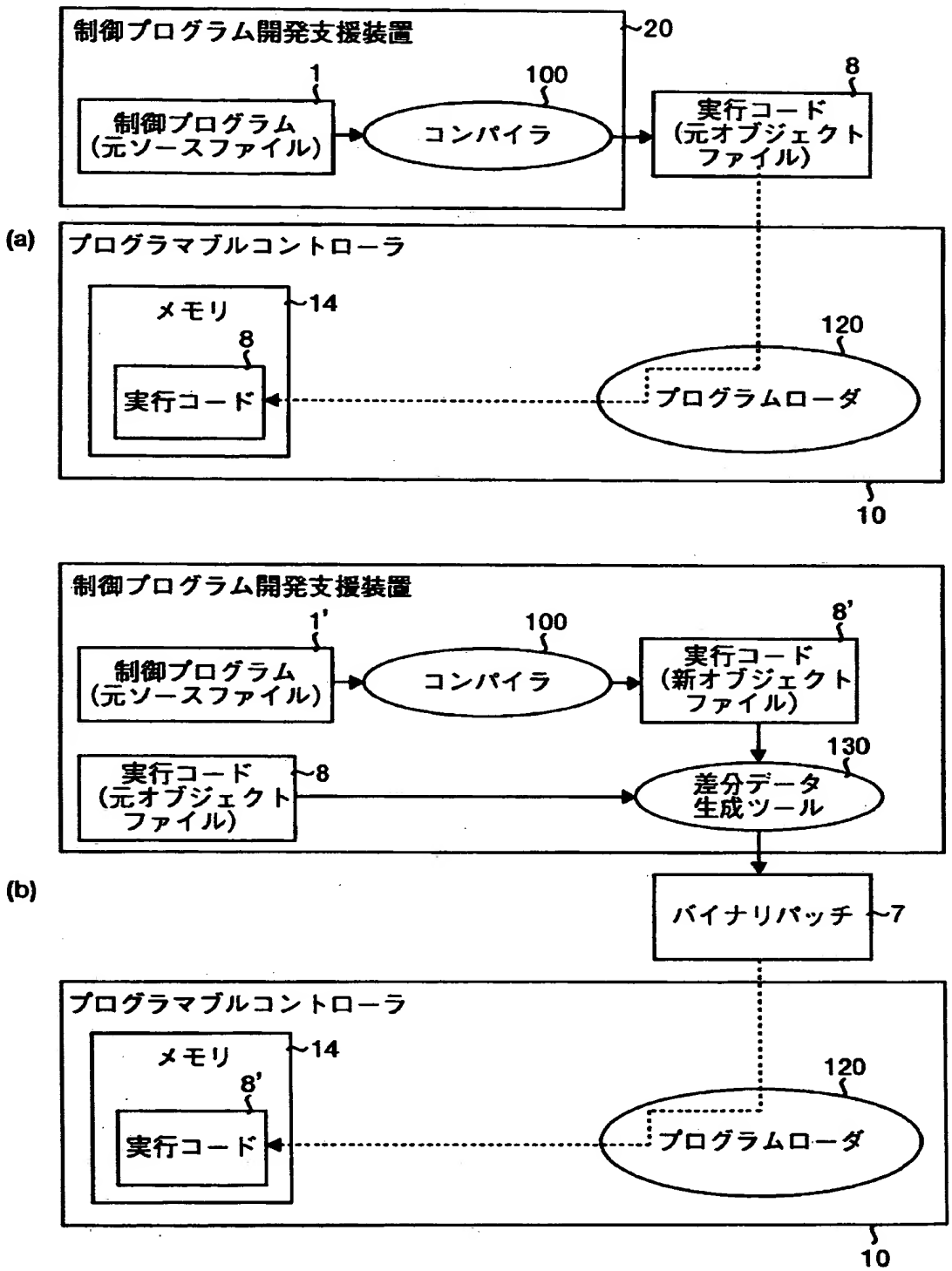
【図２０】



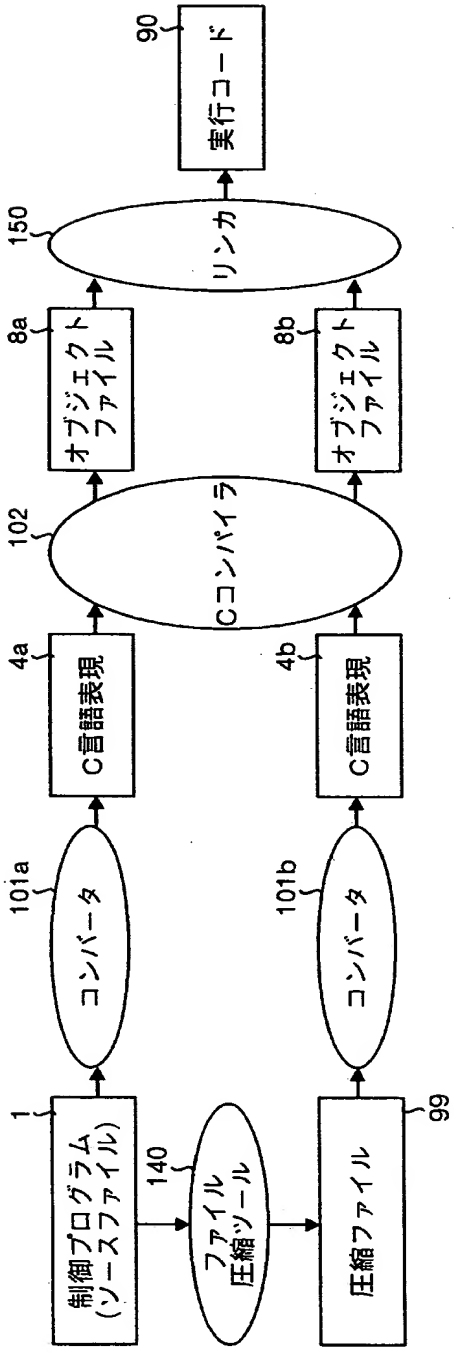
【図２１】


制御プログラム
(入力ソースファイル)
1
LD X0
AND X1
LD M0
OR M1
ANB
OUT Y0
コンバータ
101
高級言語変換ファイル
4
void main(){
while(1){
int bit0,bit1,bit2;
#line 1"入力ソースファイル.c"
bit0=X0;
#line 2"入力ソースファイル.c"
bit0 &=X1;
#line 3"入力ソースファイル.c"
bit1=M0;
#line 4"入力ソースファイル.c"
bit1 |=M1;
#line 5"入力ソースファイル.c"
bit0 &=bit1;
#line 6"入力ソースファイル.c"
Y0=bit0;
}
}
デバッガ
110
1:> LD X0
2: AND X1
3: LD M0
4: OR M1
5: ANB
6: OUT Y0
コンパイラ
108
実行コード
(オブジェクトファイル)
9

【図２２】


(a)
制御プログラム開発支援装置
20
1
制御プログラム
(元ソースファイル)
100
コンパイラ
8
実行コード
(元オブジェクト
ファイル)
プログラマブルコントローラ
メモリ
14
8
実行コード
120
プログラムローダ
10
(b)
制御プログラム開発支援装置
1'
制御プログラム
(元ソースファイル)
100
コンパイラ
8'
実行コード
(新オブジェクト
ファイル)
実行コード
(元オブジェクト
ファイル)
8
130
差分データ
生成ツール
バイナリパッチ
7
プログラマブルコントローラ
メモリ
14
8'
実行コード
120
プログラムローダ
10

【図２３】


90
実行コード
150
リンカ
8a
オブジェクトファイル
8b
オブジェクトファイル
102
Cコンパイラ
4a
C言語表現
4b
C言語表現
101a
コンパータ
101b
コンパータ
1
制御プログラム(ソースファイル)
140
ファイル圧縮ツール
99
圧縮ファイル

【図２４】

ソースファイル

```
LD      X0
OR      X2
LD      X1
OR      X3
ANB
LD      X4
AND     X5
ORB
OUT     M0
LD      X1C
MPS
AND     M8
OUT     Y30
MPP
OUT     Y31
LD      X1D
MPS
AND     M9
MPS
AND     M68
OUT     Y32
MPP
AND     T0
OUT     Y33
MPP
OUT     Y34
```

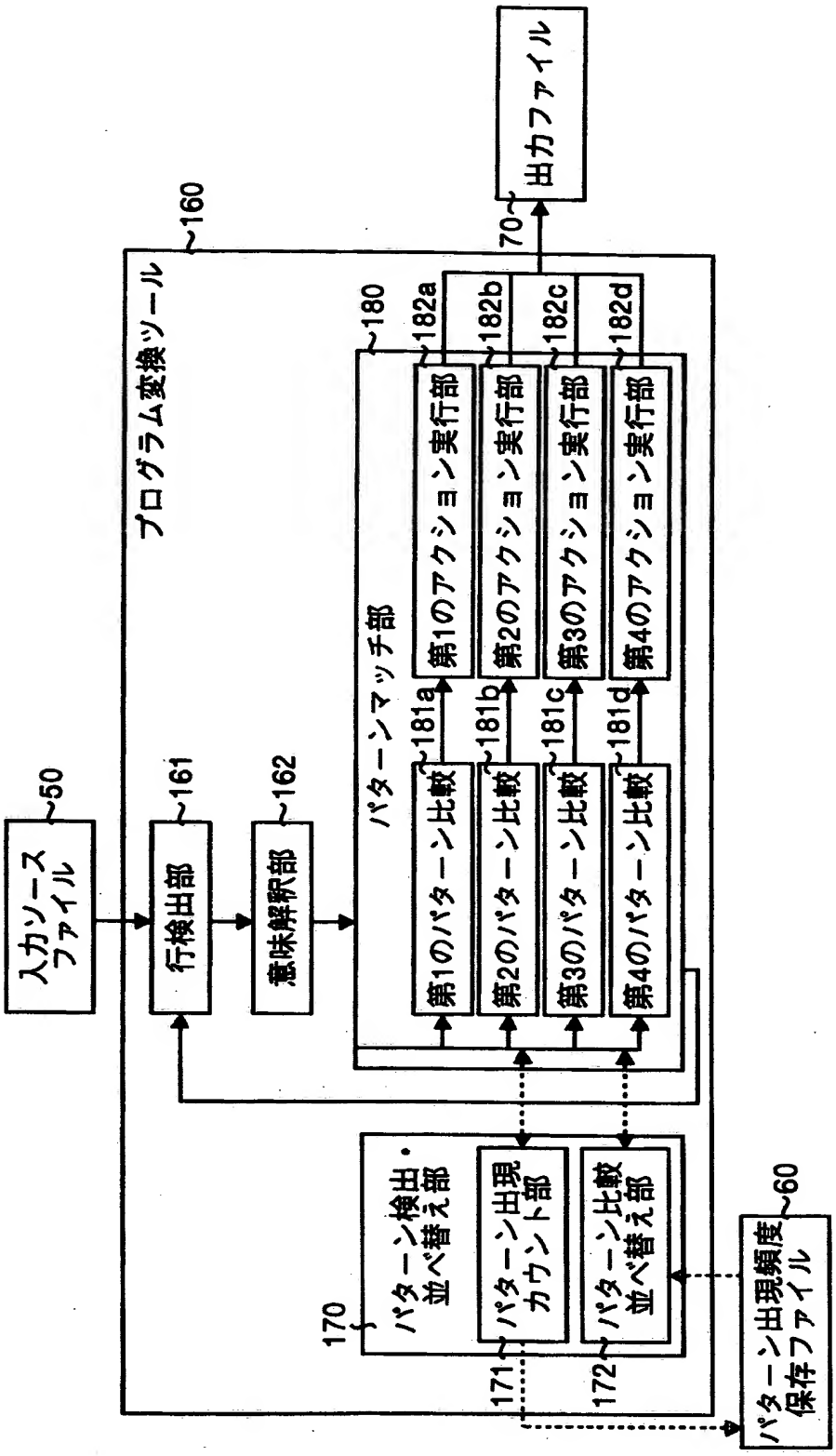
【図２５】

圧縮ファイルのC言語表現

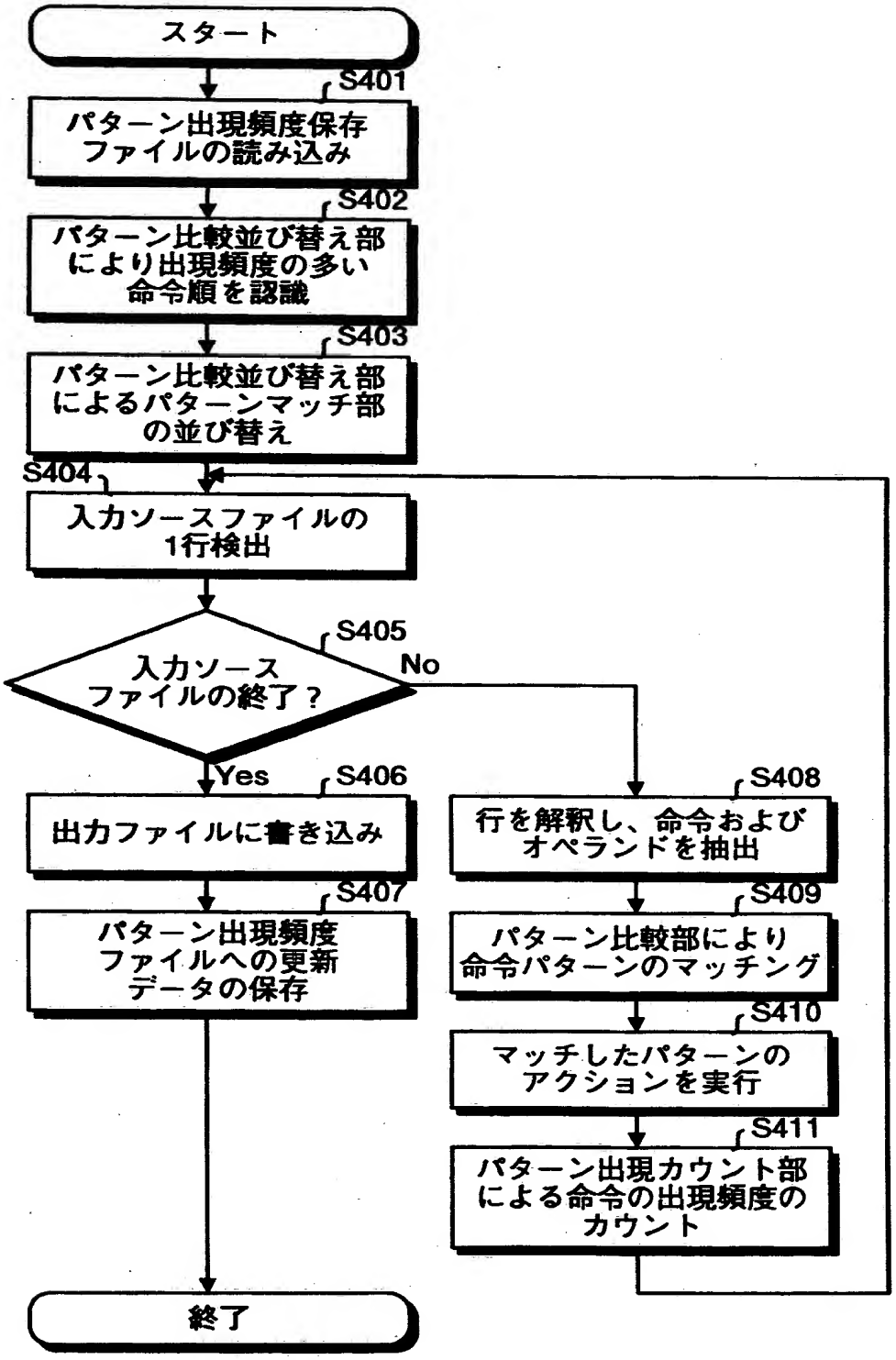
```
signed char* _InputSourceName="sample.ll.gz";

unsigned char _InputSource[]={
    0x1f,0x8b,0x08,0x08,0x9b,0xd9,0x84,0x39,0x00,0x03,0x73,0x61,0x6d,0x70,0x6c,0x65,
    0x2e,0x69,0x6c,0x00,0xf3,0x71,0xe1,0x8c,0x30,0xe0,0xe5,0xf2,0x0f,0xe2,0x8c,0x30,
    0xe2,0xe5,0xf2,0x01,0xf2,0x0c,0x21,0x3c,0x63,0x5e,0x2e,0x47,0x8f,0x27,0x88,0x90,
    0x09,0x88,0x0d,0xa4,0x4d,0x41,0x52,0x40,0x31,0xff,0xd0,0x10,0x4e,0x5f,0x03,0xa8,
    0x72,0x67,0x5e,0x2e,0xdf,0x80,0x60,0x88,0x0a,0x5f,0x0b,0x88,0x64,0xa4,0xb1,0x01,
    0x48,0x34,0x00,0xc6,0x33,0x84,0xaa,0x75,0x41,0x56,0x6b,0x89,0xcc,0x31,0x83,0xeb,
    0x34,0x82,0xea,0x04,0x09,0x87,0x18,0xc0,0x44,0x8d,0x51,0xcc,0x03,0x3a,0x08,0x00,
    0x10,0xeb,0x7d,0x3a,0xb9,0x00,0x00,0x00};
```

【図２６】


入力ソースファイル
50
行検出部
161
意味解釈部
162
プログラム変換ツール
160
パターンマッチ部
180
第1のパターン比較
181a
第2のパターン比較
181b
第3のパターン比較
181c
第4のパターン比較
181d
第1のアクション実行部
182a
第2のアクション実行部
182b
第3のアクション実行部
182c
第4のアクション実行部
182d
出力ファイル
70
パターン検出・並べ替え部
170
パターン出現カウント部
171
パターン比較並べ替え部
172
パターン出現頻度保存ファイル
60

【図２７】


スタート
S401
パターン出現頻度保存ファイルの読み込み
S402
パターン比較並び替え部により出現頻度の多い命令順を認識
S403
パターン比較並び替え部によるパターンマッチ部の並び替え
S404
入力ソースファイルの1行検出
S405
入力ソースファイルの終了？
No
Yes
S406
出力ファイルに書き込み
S407
パターン出現頻度ファイルへの更新データの保存
終了
S408
行を解釈し、命令およびオペランドを抽出
S409
パターン比較部により命令パターンのマッチング
S410
マッチしたパターンのアクションを実行
S411
パターン出現カウント部による命令の出現頻度のカウント

【書類名】　　要約書

【要約】

【課題】　より高速なシーケンス処理を実現するプログラマブルコントローラと制御プログラム開発支援装置を得ること。

【解決手段】　高速化実装技術により構築された汎用マイクロプロセッサ１１を搭載したプログラマブルコントローラ１０においてシーケンス処理を実行させるために、制御プログラム開発支援装置２０がその汎用マイクロプロセッサ１１において直接実行可能な実行コードを生成する。

【選択図】　　図１

出　願　人　履　歴　情　報

識別番号　　〔０００００６０１３〕

１．変更年月日　　１９９０年　８月２４日
［変更理由］　新規登録
住　所　東京都千代田区丸の内２丁目２番３号
氏　名　三菱電機株式会社